Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Sami Hakkarainen

Data-Driven Sequential Monte Carlo Motion Synthesis

Master's Thesis
Espoo, August 28, 2014

Supervisor: Assistant Professor Perttu Hämäläinen
Advisor: Assistant Professor Perttu Hämäläinen

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Sami Hakkarainen

Title:
Data-Driven Sequential Monte Carlo Motion Synthesis

Date:	August 28, 2014	**Pages:**	89
Major:	Media Technology	**Code:**	T-111
Supervisor:	Assistant Professor Perttu Hämäläinen		
Advisor:	Assistant Professor Perttu Hämäläinen		

Animation in video games is composed of motion segments created by animators, and of motion synthesis methods, which combine and extend the motion segments for emerging gameplay situations. Current video games typically synthesize motion kinematically with no regard to dynamics, causing immersion-breaking motion artifacts. By contrast, physically-based methods synthesize motions by simulating physics, which ensures physical correctness.

This thesis extends sequential Monte Carlo motion synthesis, a physically-based method, to use animator-authored reference animations for guiding the synthesis. An offline component is developed, which robustly tracks various types of kinematic reference animations by controlling a simulated physical character. The tracking results are gathered as a training set for a machine learning component, which directs the sequential Monte Carlo sampling used for online motion synthesis.

For machine learning, the approximate nearest neighbors, locally weighted regression, mixture of regressors, and self-organizing map methods are implemented and compared. A product distribution sampling scheme is developed to efficiently combine machine learning with optimization. Additionally, a factorized formulation of the learning problem is presented and implemented.

The system is evaluated with an interactive locomotion test case. Given a single kinematic reference animation depicting running in a straight line, the system is able to synthesize physically-valid motion for turning and running on uneven terrain.

Keywords:	motion synthesis, procedural animation, physically-based animation, machine learning, regression, Monte Carlo methods, dimensionality reduction, optimization
Language:	English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Sami Hakkarainen		
Työn nimi:	Dataohjattu sekventiaalinen Monte Carlo -liikesynteesi		
Päiväys:	28. elokuuta 2014	**Sivumäärä:**	89
Pääaine:	Mediatekniikka	**Koodi:**	T-111
Valvoja:	Apulaisprofessori Perttu Hämäläinen		
Ohjaaja:	Apulaisprofessori Perttu Hämäläinen		

Videopelien animaatio muodostuu animaattoreiden luomista animaatioista, sekä liikesynteesimenetelmistä, jotka yhdistävät ja laajentavat luotuja animaatioita pelissä syntyviin uusiin tilanteisiin. Nykyiset videopelit käyttävät pääsääntöisesti menetelmiä, jotka syntetisoivat liikettä kinemaattisesti huomioimatta dynamiikkaa, mikä johtaa immersiota heikentäviin virheisiin. Vaihtoehtoisesti liikesynteesiin voidaan käyttää fysiikkaan perustuvia menetelmiä, joissa fysiikan simuloinnilla varmistetaan liikkeiden fysikaalinen toteutettavuus.

Tämä diplomityö laajentaa fysiikkaan perustuvaa sekventiaalista Monte Carlo -liikesynteesimenetelmää ohjaamalla synteesiä animaattoreiden luomilla referenssianimaatioilla. Työssä kehitetään erillinen komponentti, joka kykenee seuraamaan monenlaisia kinemaattisia referenssianimaatioita kontrolloimalla simuloitua fysikaalista hahmomallia. Seurannan tulokset kootaan opetusdataksi koneoppimiskomponentissa, joka ohjaa interaktiiviseen liikesynteesiin käytettävää sekventiaalista Monte Carlo -otantaa.

Koneoppimiseen sovelletaan approksimatiivista lähimmän naapurin menetelmää, paikallisesti painotettua regressiota, regressorisekoitemallia ja itseorganisoituvaa karttaa. Koneoppiminen yhdistetään tehokkaasti optimointiin käyttämällä otantaa todennäköisyysjakaumien tulosta. Oppimisongelmaan sovelletaan myös tekijöihin jaettua muotoa.

Järjestelmää arvioidaan interaktiivisella demonstraatiolla, jossa käytetään yksittäistä suoraa juoksua esittävää kinemaattista referenssianimaatiota. Järjestelmä kykenee syntetisoimaan referenssin avulla käännöksiä ja juoksua epätasaisella pinnalla.

Asiasanat:	liikesynteesi, proseduraalinen animaatio, fysikaalinen animaatio, koneoppiminen, regressio, Monte Carlo -menetelmät, ulotteisuuden pienentäminen, optimointi
Kieli:	Englanti

Acknowledgements

I would like to thank professor Perttu Hämäläinen for giving me the opportunity to work on an intriguing and highly educational thesis subject. Thank you for your invaluable advice and direction.

I would also like to thank coworkers Esa Tanskanen and Sebastian Eriksson for welcoming me to the project, as well as for the helpful and entertaining discussions.

The motion capture data used in this work was performed by Klaus Förger and Perttu Hämäläinen. Both deserve a big thank you for their time and their performances. Klaus receives additional thanks for reading the thesis and providing feedback.

Finally, I would like to thank my family and friends for acting interested during my spontaneous lectures on computer animation and machine learning.

Helsinki, August 28, 2014

Sami Hakkarainen

List of symbols

$\hat{\mathbf{x}}$	State features vector
$\hat{\mathbf{y}}$	Control signal vector
$\hat{\mathbf{x}}_\mathbf{0}$	Current state features in online optimization
$\mathbf{x}$	State features after pose dimensionality reduction
$\mathbf{y}$	Control signal after pose dimensionality reduction
$\mathbf{z}$	Paired state features and control signals
$\hat{\mathbf{X}}$	State features in the training set
$\hat{\mathbf{Y}}$	Control signals in the training set
$\hat{Y}$	The space of control signals
Y	The space $\hat{Y}$ after pose dimensionality reduction
$\mathbf{T}_{Y \leftarrow \hat{Y}}$	Transformation matrix from space $\hat{Y}$ to space Y
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	Multivariate Gaussian probability distribution
$\boldsymbol{\Sigma}_{xx}$	Marginal covariance matrix of $\mathbf{x}$
$\boldsymbol{\Sigma}_{k,xx}$	Marginal covariance matrix of $\mathbf{x}$ for class k
$\boldsymbol{\Sigma}_{y\|x}$	Conditional covariance matrix of $\mathbf{y}$ given $\mathbf{x}$
λ	Regularization parameter
Δt	Simulation / optimization time step
$\mathbf{p}, \mathbf{q}$	Position vector and orientation quaternion
$\mathbf{v}, \boldsymbol{\omega}$	Linear and angular velocities
$\mathbf{S}$	Simulated motion
$f(\mathbf{S})$	Fitness function
$\mathbf{d}_{average}$	Deviation vector defining the average cost
$\mathbf{d}_{terminal}$	Deviation vector defining the terminal cost
σ	Standard deviation
$\mathbf{q}_i$	Joint angles for segment i of the spline
$\mathbf{l}_i$	Maximum torques for segment i
t_i	Time duration of the segment i
t_{min}	Shortest possible spline segment duration
t_{max}	Longest possible spline segment duration
$t_{horizon}$	Maximum time horizon length

Contents

Acknowledgements **4**

List of symbols **5**

1 Introduction **8**

2 Animation and motion synthesis **12**
2.1 Computer animation 12
2.2 Animated virtual human characters 13
2.3 Kinematics-based motion synthesis 14
2.4 Biomechanical modeling 15
2.5 Physics simulation 17
2.6 Motion synthesis as control 18
2.7 Optimizing controller parameters 19
2.8 Optimizing for instant objectives 20
2.9 Spacetime optimization 20
2.10 Trajectory optimization 21

3 Sequential Monte Carlo motion synthesis **23**
3.1 Stochastic optimization 23
3.2 Sequential Monte Carlo 24
3.3 Online motion synthesis using sequential Monte Carlo 26
3.3.1 Adaptive importance sampling using a kd-tree 26
3.3.2 Sequential kd-tree sampling 28
3.3.3 Parametrization 28
3.3.4 Fitness function 30

4 Data-driven sequential Monte Carlo motion synthesis **32**
4.1 System overview 32
4.2 Simulated physical character 34
4.3 Preprocessing reference animations 36

4.4 Locomotion test case . . . 37
4.5 Summary . . . 39

5 Tracking reference animations **40**
5.1 Overview . . . 40
5.1.1 Optimization . . . 41
5.1.2 Reference data . . . 42
5.2 Fitness function . . . 43
5.2.1 Average cost . . . 43
5.2.2 Terminal cost . . . 44
5.2.3 Analysis . . . 44
5.3 Sample generation . . . 45
5.4 Summary . . . 47

6 Learning control from examples **49**
6.1 Overview . . . 49
6.1.1 State features . . . 51
6.1.2 Pose dimensionality reduction . . . 52
6.2 Approximate nearest neighbors . . . 53
6.3 Locally weighted regression . . . 54
6.3.1 Model . . . 54
6.3.2 Sampling . . . 56
6.4 Mixture of regressors . . . 59
6.4.1 Model . . . 59
6.4.2 Training . . . 60
6.4.3 Sampling . . . 62
6.4.4 Factorized mixture of regressors . . . 63
6.5 Self-organizing map . . . 64
6.6 Summary . . . 68

7 Evaluation **69**
7.1 Offline component . . . 69
7.2 Online component . . . 70
7.2.1 Robustness . . . 73
7.2.2 Performance and sample counts . . . 75
7.2.3 Motion quality . . . 77
7.3 Analysis . . . 78

8 Conclusions and future work **80**
8.1 Conclusions . . . 80
8.2 Future work . . . 81

Chapter 1

Introduction

The field of computer animation has matured rapidly and is revolutionizing film production. Increased computing power and advances in rendering technology have driven the look of computer-animated films forward towards increasingly realistic-looking characters. The movement of the characters in these films is now also believable, thanks to motion capture technology and the skillful work of animators.

While current animation technology allows film animators to share their work unaltered, video game animators must deal with the challenges caused by interactivity. Interactivity allows the player to experience an infinite number of situations, all of which would require convincing animations to keep the player immersed in the game. The situations are also overlapping in time: a video game character might have to, for example, dodge a punch, walk forward, and taunt the opponent all at the same time depending on the player's input. Video game animators must also respect the requirement of responsiveness, that is, they must make the character respond to the player's input without delay.

Because of these requirements, even the best current video games cannot guarantee believable motion at all times. Video game characters often gain or lose momentum without explanation, neglect to respond to a contact, or move around in a repetitive manner. Furthermore, the design of video games is limited by the difficulties of animation.

The problem of creating convincing animation for the infinite emerging situations has been studied in the field of motion synthesis. A couple of successful approaches exist. In kinematics-based motion synthesis, motion segments authored by the animator are concatenated and blended in real-time to synthesize new motions. For example, a walking motion may be concatenated and blended with a running motion to create a controller for varying speeds. The animator may easily change the style of the motions by

editing the original motion segments, resulting in high-quality animation.

The main disadvantage of this approach is that kinematic motion synthesis gives no consideration to the causes of the motion. Synthesized motions may not be physically feasible, even if the original motion segments are. Furthermore, the character only responds to events if the animator has created a suitable motion segment for the specific case. For example, the character responds to pushes only in the ways the animator has specified.

Physically-based motion synthesis is an alternative to the kinematics-based approach. The character is modeled as a set of physical objects connected by joints, which are then simulated forward in time to synthesize motions. This way, all motions of the character are naturally constrained to be physically feasible. This approach has had the most applications in the simulation of unactuated (unpowered) motion, which is known informally as ragdoll physics.

Motion synthesis can also be achieved with actuated physical characters, i.e. characters with motored joints. This type of motion synthesis is still largely an open research problem, but if solved, actuated characters could be used to generate animator-authored physically feasible motion. The general problem is to generate a suitable control signal for the character's actuators, given the current state and the objective. Researchers have approached the problem through many disciplines, such as control theory, biomechanics, and machine learning.

Recent research has created convincing results with the use of trajectory optimization. Instead of only inspecting the immediate situation, trajectory optimization generates a control signal for a short window forward in time. The signal is created in such a way that it is optimal with respect to a measure known as the fitness (or cost) function, which gives high-level direction to the character, for example, to move forward or to stay still in a given pose. Unfortunately, the optimization problem is computationally demanding to solve, which limits the use of trajectory optimization in interactive applications. Even seemingly simple tasks, such as following a trajectory given by a kinematic reference animation, are difficult to achieve.

One recent and interesting type of trajectory optimization is called sequential Monte Carlo motion synthesis [32]. Like other Monte Carlo optimization methods, it employs random sampling to find an answer to the optimization problem. It generates multiple random control signal samples and plays them forward in time using the simulated physical character, yielding multiple realized motions. The control signal samples are then weighted by evaluating the fitness function for the corresponding realized motions. Samples with high fitness are then used to guide the generation of new random samples, focusing the process towards optimal control signals.

In the simplest type of sample generation, every random control signal in the vast optimization space is initially equally likely. The optimization process is given no additional information on how the task should be performed, that is, the motion is synthesized from scratch, and no motion data created by an animator is needed. The effectiveness of this type of sequential Monte Carlo motion synthesis has been demonstrated for some tasks, namely getting up and balancing in a standing pose. Robust control of the character is achievable at interactive frame rates for these types of tasks.

While this type of motion synthesis is already useful and interesting, it is further extendable by incorporating prior knowledge in the form of animator-authored motion data, consisting of keyframed or motion-captured animations. For example, instead of letting the optimizer synthesize a completely novel getting up motion, the animator could give the optimizer examples of a specific style.

The potential advantages of using prior knowledge are two-fold. First, the quality of the synthesized motions should be improved through the increased control given to the animator. Second, for some motion tasks, prior knowledge should be able to drastically improve the optimization efficiency by directing the sampling to only the types of control signals that are known to be useful. For example, instead of exploring the vast space of all the possible control signals a human can perform, the sampling could be focused on the tiny portion that yields walking motions.

This thesis aims to investigate how this type of prior knowledge could be most efficiently incorporated into sequential Monte Carlo motion synthesis. We aim to gain the two potential advantages explained above - efficiency and quality - without restricting the applicability to novel situations. The resulting framework is called *data-driven sequential Monte Carlo motion synthesis*.

The field of motion synthesis is relatively new, but nevertheless large and inherently multidisciplinary. Therefore, this thesis must be delimited to discuss only some aspects of the field. First, accurate simulation of biomechanics is well-studied and crucial for synthesizing natural motions, but our model of the human biomechanics is very simple. Second, various optimization and control methods have been successfully applied to motion synthesis, but this thesis is limited to only discuss the sequential Monte Carlo framework. Last, although the goal is to create a framework that is applicable to all types of motion tasks, we focus primarily on locomotion.

The contribution of this thesis consists of three parts. First, we apply the sequential Monte Carlo framework to track kinematic reference animations of various kinds. Then, we implement and compare a number of machine learning methods that direct the online Monte Carlo optimization process with the help of reference motions learned offline. Last, we present an online

locomotion test case as an application of the data-driven sequential Monte Carlo motion synthesis framework.

In Chapter 2, we give a brief introduction to related work in the field of motion synthesis. Next, in Chapter 3, we present the sequential Monte Carlo optimization framework, which is the basis from which this thesis is extended. In Chapter 4, we overview the main contribution of this thesis: the data-driven sequential Monte Carlo motion synthesis framework. We also present the locomotion test case as a concrete example of the framework in use. The next chapters explain the parts of the implemented system in detail: Chapter 5 discusses the reference animation tracking and Chapter 6 presents the machine learning methods. In Chapter 7, we present the results of our experiments. Finally, in Chapter 8, we conclude the thesis and map out future work.

Chapter 2

Animation and motion synthesis

In this chapter - before going into the details of sequential Monte Carlo motion synthesis - we will explain related work in the field. We will start by explaining some basic concepts and terms of computer animation.

The terms *animation* and *motion synthesis* both refer to roughly the same thing: the creation of motion. We distinguish between the two by using the term animation to denote the creation of motion by humans and reserve the term motion synthesis for algorithmic creation of motion.

2.1 Computer animation

Before interactive simulations were possible, animation meant strictly the creation of static images (frames) that were shown rapidly in sequence to create the illusion of motion. In hand-drawn animation, or traditional animation, every frame of an animation had to be penciled in by hand. In most film productions, this meant that 24 drawings had to be created for every second of finished animation - a huge pile of drawings for a full-length movie.

The laborious task was commonly split between a senior and a junior animator: the senior animator, called key animator, would draw the key poses that defined the major points of the motion, while the junior animator would have the easier job of filling in the gaps. In today's computer-assisted animation every animator is essentially a key animator, as the drawing and the filling of the gaps is left to the computer. [71]

The animated frames are drawn using rendering, which is the process that transforms the 3D geometry and the materials of the scene into a 2D image. The rendering methods used in modern animated films are physically-based: they aim to correctly model the scene using the theory of light's behavior on various materials. We will not discuss rendering further in this thesis,

instead, we refer to the often cited textbooks in online and offline rendering. [59] [3]

We cannot avoid the topic of motion quality in this thesis. Though the perceived quality is obviously subjective, there are some commonly accepted approaches to the matter. The creation of principles on the quality of animation is attributed to the animators at the Walt Disney Studio in the 1920's and 1930's, long before the field of computer animation emerged. The principles created - such as squash and stretch, anticipation, and overlapping action - have survived digitalization and are now commonly taught to students of computer animation. [45]

Although the quality of motion is not entirely defined by physical realism, the physically-based motion synthesis methods discussed in this thesis have been shown to adhere well to the principles of animation. In particular, the synthesized motions preserve a sense of weight because of the physical constraints. [72]

2.2 Animated virtual human characters

Although the term animation is usually associated only with translations and rotations of objects, modern animators animate everything in the scene: shapes, lights, colors, etc. Many types of objects and phenomena can be animated: stacks of boxes falling, oceans waving, characters interacting, etc. The focus of this thesis is on a specific but common subset: the animation of human characters.

Instead of remodeling the character's geometry for every animated frame, the animator poses the character by manipulating a small number of controls. In skeletal animation, the characters have a hierarchy of controls to rotate every major bone in the body. The orientation of a single bone can be efficiently represented with three relative angles at the joint connected to the bone and orientations of the parent bones in the hierarchy.

The way the character's geometry moves in relation to the animated bones is defined by a skinning algorithm. In real-time rendering, linear blend skinning and dual quaternion skinning [38] are the commonly used techniques.

In keyframe animation, virtual characters are animated by manually posing the character in keyframes and letting the computer interpolate the motion in between. With keyframe animation, the animator can create any type of motion for any type of character. However, creating believable motion by keyframing requires a lot of effort and skill.

In motion capture animation, the movements of a set of markers placed on a real human performer are recorded using a sensor system. The recorded

movements of the markers are then retargeted onto the virtual character's skeleton. Although the use of motion capture is generally limited to the types of motion performable by a human actor, it has become prevalent after affordable commercial motion capture systems have become available.

2.3 Kinematics-based motion synthesis

Next, we move the discussion from animating to algorithms that synthesize motion in interactive applications. The most successful approach is the use of kinematics-based methods, sometimes called example-based motion synthesis. Their use is prevalent in applications: the majority of the motion in big video game franchises such as Grand Theft Auto or Assassin's Creed is synthesized using these methods. [57]

Kinematics-based methods consist of two basic building blocks: concatenation and blending. Concatenation simply means playing one motion segment after another. In blending, two or more similar motions are interpolated to create a new motion. A set of parameters called blending weights govern how much of each motion is mixed into the blend. Skeletal animation is particularly easy to blend, as interpolation can be calculated separately for each joint in the hierarchy.

Large sets of motion segments are commonly organized into a data structure called motion graph [41], in which the edges of the graph represent motion segments, while the nodes of the graph represent choice points, at which two motion segments can be seamlessly concatenated. Parametric motion graphs [30] are a common extension to motion graphs in which blending is used to create a continuous motion space.

Since manually constructing a motion graph is tedious and difficult, much of the research in kinematic-based motion synthesis is focused on the automatic generation of motion graphs. Various distance metrics have been proposed to find suitable transition points between different motions, and various strategies have been used to synthesize the transition motion.

After the motion segments have been organized in the motion graph, new motion can be synthesized by visiting the edges of the graph in some order and playing the corresponding motion segments. In an interactive application, a character controller selects the edges to visit using some measure of optimality. The most common method searches for optimal motions only in the local neighborhood of the current node. The obvious disadvantage is that the motion synthesized using only the local information may not be globally optimal.

Synthesizing globally optimal sequences of motion would allow for in-

creased quality and more complex motion sequences. For example, characters could have complex interactions with the environment or with other characters. The problem of synthesizing motion in this way is similar to the motion planning problem in robotics, which has lead to similar approaches being used for both problems. For example, reinforcement learning has been recently used in motion synthesis. [57]

Although the vast majority of interactive applications use kinematics-based motion synthesis, there are some limitations. For example, the result of blending two dissimilar motions is not guaranteed to be physically correct. Most of the incorrectness goes unnoticed by the viewer, but in some cases the problems resulting from blending are easy to notice, for example when the feet supporting the character slide along the ground. Some of the problems may be alleviated with additional logic, for example by using inverse kinematics (IK) to lock the sliding feet in place [42].

At best, the results of simple interpolations of some motions are nearly physically correct [61], but in practice creating a complete interactive simulation which maintains physical correctness is extremely difficult. Well-animated interactive applications such as video games require a large collection of carefully crafted rules and scripts that choose which motion segments are selected to synthesize the final motion. In addition to the complex logic, video games requires thousands of short motion segments to be animated even for a relatively small set of character abilities. [22]

2.4 Biomechanical modeling

To move from the kinematics-based motion synthesis to physically-based methods, the previously introduced skeletal animation model needs to be augmented with physical properties. For this, we briefly explain relevant information in biomechanics, which is the field that studies the structure and function of biological systems, such as humans. We also explain how biomechanics is commonly approximated in physically-based motion synthesis.

The geometry of the character is typically modeled only as a hierarchy of bones, ignoring the soft tissue and other complexities. The bones are modeled as inelastic and incompressible rigid bodies. Some applications use a detailed triangle mesh encompassing the skin as the bone's collision geometry, but most use simplified shapes such as capsules or boxes. The mass distribution and density of the character may be modeled after real-life data, though many simply assume a constant density.

Although the joints in humans are complicated structures with more functionality than simply connecting the bones together, the simulated joints are

commonly modeled as hard constraints which inelastically connect the bones. The character's joints are commonly modeled as hinge joints, which have 1 degree-of-freedom (DOF), and as ball-and-socket joints, which have 3 DOF. The movement range of the joints is usually limited to approximate normal human capabilities.

For some applications, there is no need to model every degree of freedom a real human would have. For example, ankle joints are often modeled as hinge joints, as the added range of motion is unnecessary for most motion tasks. Some applications may even model the whole upper body as a single rigid body, if the the application is focused on the simulation of the legs.

Humans generate torque by using muscles which connect to bones through tendons. Most muscles generate torque around only one joint, but some span over two. The amount of torque generated by the muscles depends on many factors, one of which is the pose-dependent moment arm of the muscle. Other factors include the dependence on the length and velocity of the muscle fibers.

Most physically-based motion synthesis research uses only simple servo-based actuation, in which muscles and tendons are not modeled, but a single motor is assigned for each joint. The motors typically have a constant maximum torque, which is often adjusted per joint. The servo model is common, since it is simple to implement, and since posing the character with the target angles is more intuitive than in other methods. [22]

Humans control the use of muscles using their motor system: a large portion of the central nervous system dedicated for motor control. Research in physically-based motion synthesis is typically concerned in modeling the higher level control housed in the cerebral cortex, while other important parts of the complex motor system are neglected. For example, in real humans, the sensory input received from the muscles, tendons, and the skin is used by lower parts of the motor system as a form of closed-loop control. [9]

Because of efficiency concerns, the biomechanical models commonly used in physically-based motion synthesis are extremely simple compared to the real-life counterpart. However, some authors have developed more accurate biomechanical models with great results. For example, Jain and Liu [36] improve the robustness and motion quality of relatively simple controllers using simulated deformable soft tissue. The simulation of muscles and tendons by Wang et al. [70] and later by Geijtenbeek et al. [24] greatly improves the quality of the generated motions.

While accurate modeling of biomechanics is relatively new in motion synthesis, biomechanics scientists have developed fine-grained simulations for medical research. For example, the open source OpenSim [16] system allows for accurate analysis and simulation of the human musculoskeletal system.

2.5 Physics simulation

Next, we will briefly describe how the physical model of the character can be transformed into motion using a physics simulator. The numerical simulation of physics is a well-studied field: several different types of phenomena can be simulated, some efficiently enough to be used in interactive applications. For example, video games can simulate collapsing buildings, clothing, large scale water effects, etc.

This thesis deals only with the simulation of human characters. Thus, we discuss only the type of rigid body simulation typically used for this task, although other phenomena could be simulated using the same general approach.

In typical rigid body simulations, time is discretized into frames of constant length Δt. The following steps are taken each frame:

1. Forward dynamics. Computes the linear and angular accelerations for the rigid bodies based on forces and torques.

2. Numerical integration. Computes the linear and angular velocities and positions based on the accelerations.

3. Collision detection. Finds all intersecting rigid bodies and computes additional contact information such as contact normals.

4. Collision response and constraint handling. Resolves the collisions so that no rigid bodies are intersecting, enforces additional constraints.

The first two steps are relatively easy to implement and efficient to compute. The majority of the computational effort is spent on the remaining two steps.

Collision detection is a well-studied problem for which applications exist outside physics simulation. The general idea in solving the problem efficiently is to subdivide the space to avoid calculating intersection tests between every pair, which would scale as $O(n^2)$ for n objects. Instead of discussing the problem further, we refer to relevant literature [21].

The collision response and constraint handling step is the most interesting one in the context of simulating human characters, as there are some relevant design choices. The problem can be approached mathematically by modeling the collisions and constraints as a linear complementarity problem (LCP). [7] [5]

In video games, the problem is usually solved using an iterative method, such as Gauss-Seidel iteration, which resolves the constraints one at a time. The iterative method works well, since in typical situations in games there

are few collisions and constraints to solve, the number of interacting bodies is small, and the accuracy of the result is less important than the performance.

However, when simulating actuated characters supported by a long hierarchy of bones, the iterative solving methods are less efficient than direct solving methods, such as the ones based on Dantzig's simplex algorithm [14].

Implementing a robust and efficient physics simulator is a time-consuming and difficult task. Fortunately, multiple physics engines that fit the description are available for integration. For example, Open Dynamics Engine (ODE) and Bullet are popular open source engines used in many applications, and PhysX and Havok are commercial engines often used in video games. In the past, the available engines have typically only implemented iterative solvers, but direct solvers are becoming more widespread.

2.6 Motion synthesis as control

The goal of actuated physically-based motion synthesis is to control the character at all times using only torques generated by the actuators. The problem of controlling the character is difficult, since even a simple model of the human biomechanics includes a high number of actuated degrees of freedom, which must be used in cooperation to support and balance the character. Because of this difficulty, the use of physically-based motion synthesis for characters in video games and other interactive applications has been commonly limited to unactuated simulation, or "ragdolls".

Video games typically implement a system with two separate modes: a kinematics-based mode and a ragdoll mode. The switch from kinematics to ragdolls is initiated when the ragdoll-like appearance is acceptable, for example when an explosion throws the character in the air. Switching back from ragdolls to kinematics is less common because of its difficulty, but some proven approaches exist [75]. Some authors have also suggested ways in which kinematic motion can be modified in real-time for increased physical plausibility, without restricting the motion to complete physical correctness [46].

Methods from the field of control theory can be used to solve the control problem. A simple control approach which has attracted a lot of research is called joint-space motion control, in which simple PD-controllers (proportional-derivative) added to each joint control the generated torques to match some previously defined target angle. The target angles may be defined entirely procedurally or by using a data-driven approach, in which kinematic data is used to generate a target trajectory.

A kinematic trajectory cannot be robustly tracked simply by feeding the

joint angles from the kinematic data to the PD-controller target angles, as the exact physical model that created the kinematic data cannot be simulated. Even a slight difference, for example a slightly longer leg bone, will cause the character to trip and lose balance. Furthermore, even if the original conditions could be replicated, simply following the target angles would not allow the character to adapt the motion to different environments.

Some authors have proposed joint-space controllers that are able to track kinematic trajectories in some cases. For example, Sok et al. [65] precompute a time-varying displacement for the original target angles, which allows the tracking of some motion types on a 2D character. Liu et al. [48] similarly precompute displacements for a 3D character. Their system is able to track various motions, but with severely limited generalization to new environments and with long computation times.

Some researchers have developed systems that synthesize motion from scratch, i.e. without prior motion data. Various parameterizations can be used, one of which is the use of pose-control graphs, which define motion using a small number of consecutive kinematic target poses. For example, the Simbicon framework [74] uses only 2-4 poses to synthesize simple walking motions. Though pose-control graphs allow animator control in a way that is similar to kinematic keyframe animation, the quality of the synthesized motion and the robustness of the system is generally poor.

A variant of joint-space motion control called state-action mapping defines the target poses based on the character's current state. For example, the system developed by Sharon and Van de Panne [63] finds the best match in a precomputed set of target poses based on the character's joint angles, position, and orientation.

2.7 Optimizing controller parameters

In the remainder of this chapter, we will look at the various ways numerical optimization is used in physically-based motion synthesis. As the first example, we will describe how some authors have used offline optimization to find parameters for online controllers.

An early example of this is the virtual creatures system developed by Sims [64], which uses a genetic algorithm to create novel neural network control systems and morphologies for simple physical creatures. The system presented generates interesting and complex movements for simple creatures with low amounts of degrees of freedom, but the results have not been extended to characters with high degrees of freedom. [22]

A more recent example is the work of Wang et al. [70], who use co-

variance matrix adaptation evolution strategy (CMA-ES) [28] to optimize a pose-control graph controller for minimal usage of metabolic energy. The synthesized motions look natural and the gait well matches ground truth data recorded from actual human motion. The major disadvantage of their system is the limited generalization to new environments.

In addition to motion synthesis from scratch, parameter optimization has also been used to improve controllers tracking kinematic trajectories. Geijtenbeek et al. [23] use CMA-ES to optimize both their PD-control gains and the weights used for their separate balance controller. For some movements, their tracking produces high quality motion at interactive frame rates, but their method does not work for locomotion tasks, nor does it allow for large disturbances.

2.8 Optimizing for instant objectives

Numerical optimization can also be used online to find optimal control signals. Compared with the simple joint-space control, the robustness of control is increased, as the character's actuators can be used in cooperation.

Abe et al. [1] balance the character by optimizing using quadratic programming (QP). Their fitness function defines balance as the position of the center of mass in relation to the base of support. Similarly, Jain et al. [37] use QP to optimize for one incremental balancing objective at a time by moving the end effectors to supporting positions.

While some of the results of controller optimization produce good quality motions at interactive frame rates, all the methods presented so far offer limited generalization to new situations. For example, when using the simple balance controllers, balance is maintained only if the next incremental objective is within reach: after losing balance, the balance controllers cannot generate the complex motions that return the character back to a stable state.

2.9 Spacetime optimization

So far, we have looked at offline optimization for controller parameters and online optimization of control for instantaneous objectives. A third approach to use optimization for motion synthesis is to optimize the resulting motion over time.

The spacetime constraints formulation by Witkin and Kass [72] introduced the approach to motion synthesis. In their paper, motion synthesis is

posed as a quadratic programming problem, in which the optimized quantity is the minimization of expended energy, and a set of dynamics constraints restricts the results to physically valid motions. Additional constraints may be defined for directing the motion, for example by constraining the start and end pose of the character.

The original spacetime formulation has been refined by many authors in an effort to make the optimization problem easier to solve computationally. As an example of recent capabilities, Mordatch et al. [52] use L-BFGS to optimize for various complex motion sequences. The simulated characters interact with objects and each other using only simple high-level objectives.

The major disadvantage of this type of motion synthesis is the poor applicability to interactive use. Even the refined and simplified methods are slow: short motion sequences are synthesized in minutes or even hours. Additionally, the problem is formulated so that the motion is optimized for the full time window, which is wasteful in an interactive application, since the situation may change each frame and invalidate the optimized result for subsequent frames.

2.10 Trajectory optimization

Spacetime optimization optimizes variables which define the resulting motion and uses constraints to make the motion physically valid. Alternatively, the optimization problem can be posed as finding the optimal control signal variables using some form of prediction over time, and then implementing the control using physics simulation to constrain for physical validity. In this thesis, we call the approach trajectory optimization, though some authors prefer the related term model-predictive control (MPC).

Trajectory optimization increases robustness when tracking kinematic trajectories, as shown by da Silva et al [15]. Their work uses optimization for computing optimal torques for a short time horizon forward and PD-control for following the optimal torques. Their system runs at interactive frame rates, but offers stiff responses to external disturbances, generalizes poorly to new environments, and does not work robustly with all types of motions.

Muico et al. [53] produce high quality motion by precomputing reference motions which are then tracked using a prediction model that includes ground reaction forces. Their system synthesizes agile locomotion, such as 180 degree turns, though with high latency when initiating the turns. Extended work [54] increases robustness to external disturbances using a composite of multiple parallel controllers.

Many authors have introduced low-dimensional prediction models that

are efficient to compute even for a long time horizon. Kwon and Hodgins [44] model the character as an inverted pendulum to extend a tracked reference motion to new steering angles and speeds. Han et al. [27] use a simplified dynamics model considering the momentum at the center of mass and the changing support from foot contacts to compute predictions, which are then used as feedback for a full-body balance controller. The synthesized results are agile and robust against disturbances, though the system only works on flat terrain. Liu et al. [47] use a high-level planning component together with low-level control to synthesize parkour-style running and jumping motion sequences in real-time, though their system has severe limitations in robustness.

In addition to kinematic trajectory tracking, trajectory optimization has been used for from scratch motion synthesis. Mordatch et al. [51] synthesize locomotion using a low-dimensional foot contact model for prediction. Their method works robustly in challenging terrain, though only at 15% of real-time. Wu and Popovic [73] plan paths for end-effectors and achieve interactive navigation in uneven terrain.

Although low-dimensional models have produced good results for locomotion tasks, some complex tasks require the full high-dimensional character model. Al Borno et al. [4] use CMA-ES to optimize long motion sequences, such as getting up from the ground, and agile acrobatic motions, such as handspins. Unfortunately their method is not applicable to interactive applications, as hours of computation are required for synthesizing the motions.

Trajectory optimization using a full character model has been developed also by Tassa et al. [67], who are able to synthesize getting up motions at frame rates 7 times slower than real-time. However, as their character model is able to use supernatural control torques, the motions synthesized are quick bounces rather than natural sequences where the hands are used for support. Erez et al. [20] extend their work by dividing the motions into simple subtasks, which allows for more complex sequences to be synthesized in real-time.

Hämäläinen et al. [32] also synthesize getting up motions at interactive frame rates using trajectory optimization and a high-dimensional character model. Their work is the first to synthesize creative getting up motions from scratch without precomputation or breaking down the motion into simple subtasks.

Chapter 3

Sequential Monte Carlo motion synthesis

In this chapter, we present sequential Monte Carlo (SMC) and its application to trajectory optimization by Hämäläinen et al. [32]. We present the topic in some detail, as the main contribution of this thesis - the data-driven sequential Monte Carlo framework - is an extension of this earlier work.

We start off with some related stochastic optimization concepts. Then, we move on to SMC methods in general, and explain the specific SMC sampling algorithm used in the earlier work. Finally, we explain how the trajectory optimization problem can be solved using the SMC sampling algorithm.

3.1 Stochastic optimization

In trajectory optimization, the goal is to find a control signal $\hat{\mathbf{y}}$ that is optimal with respect to some fitness function $f(\hat{\mathbf{y}})$. Finding $\hat{\mathbf{y}}$ is not easy: the fitness function is typically non-convex, highly multi-modal, and discontinuous because of contacts.

When optimizing for control signals using black-box forward dynamics simulation, derivatives are not available, so the only way to get information about the function landscape is through costly evaluations of the fitness function at sample points. Trajectory optimization problems are very high-dimensional, so simple exhaustive searches through the parameter space, for example through grid search, are ruled out.

Difficult global optimization problems have been approached using stochastic methods, which are iterative search methods that direct the search using randomness. Rather than resorting to random walks or uniform sampling in the parameter space, various sophisticated stochastic optimization meth-

ods, that sample intelligently by assuming some degree of smoothness of the fitness function, can be used.

One example of such methods is the class of evolutionary algorithms, which includes biologically inspired algorithms for various purposes. For example, swarm algorithms, such as ant colony optimization [18], have been applied to problems presentable as graphs, while genetic algorithms [25] have been applied to discrete optimization problems.

Covariance matrix adaptation evolution strategy (CMA-ES) [28] is a stochastic derivative-free method for global optimization, which has been applied to offline trajectory optimization [4] and other motion synthesis optimization problems [70] [23]. CMA-ES is an evolutionary algorithm, i.e. it optimizes using a population of candidates solutions, which are recombinated, mutated, and selected at each iteration.

The recombination step is performed by sampling from a multivariate Gaussian distribution that represents the population. The principles used in CMA-ES are similar to the sequential Monte Carlo method described next in this chapter, though there are some key differences. First, SMC represents the population with a multimodal representation instead of a single Gaussian. Second, in CMA-ES, the distribution is re-evaluated after λ samples have been sampled and evaluated, but in the SMC algorithm described, every evaluated sample immediately affects the distribution used for drawing the next sample.

Stochastic optimization has also been approached from a purely probabilistic perspective. Bayesian optimization [50] places a probability distribution on the parameter space, and updates the distribution using information available from the evaluation of generated samples, which are in turn generated using the updated probability distribution. Using and updating the probability distribution allows a principled trade-off between exploration (sampling areas with high uncertainty) and exploitation (sampling areas with high fitness).

3.2 Sequential Monte Carlo

The methods introduced in the previous section exploit the local spatial smoothness of the fitness function for efficiency. In trajectory optimization, and in other problems with smoothly time-varying fitness functions, temporal coherence may be exploited as well. Before explaining the details, we explain some related methods and clarify terminology.

Particle filtering [26] is a method for estimating latent non-linear real-valued time-varying functions using a sequence of observations. The proba-

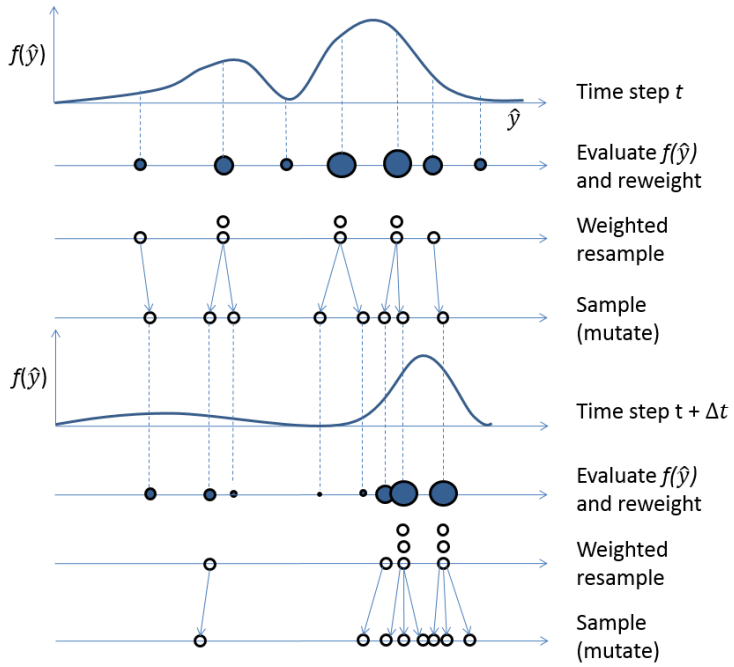


Figure 3.1: Two time-steps of a particle-based sequential Monte Carlo algorithm (Hämäläinen et al. [32]).

bilistic model used in particle filtering is closely related to Kalman filtering, which assumes Gaussian distributions, and to hidden Markov models, in which the distribution is discrete rather than real-valued. The models of this class are sometimes called state-space models (SSM), though some authors confusingly call them all hidden Markov models.

Even more confusingly, some authors regard SMC as synonymous to particle filters [26], but others consider SMC to encompass a larger set of methods. Instead of defining SMC as a method which uses a density $p(\hat{\mathbf{y}}|\mathbf{z})$ for observations $\mathbf{z}$ and latent states $\hat{\mathbf{y}}$ as in particle filtering, we define the method only in terms of a fitness function $f(\hat{\mathbf{y}})$, following the approach of Doucet and Johansen [19].

Figure 3.2 shows the three steps of a basic SMC algorithm using sequential importance resampling (SIR). The multimodal fitness function is estimated non-parametrically using a population of N samples (particles) $\hat{\mathbf{y}}_i$ in the parameter space. At each time step, the samples are first evaluated using the fitness function $f(\hat{\mathbf{y}})$. The next step is resampling: a new set of samples is generated by randomly selecting from the old set of samples using selection

weights proportional to

$$w_i \propto \frac{f(\hat{\mathbf{y}}_i)}{q(\hat{\mathbf{y}}_i)} p(\hat{\mathbf{y}}_i), \tag{3.1}$$

where $q(\hat{\mathbf{y}})$ is the proposal density from which the samples are drawn, and $p(\hat{\mathbf{y}})$ is a prior. Finally, the samples are mutated by applying noise and a prediction for the next time step.

3.3 Online motion synthesis using sequential Monte Carlo

In the remainder of this chapter, we summarize the paper by Hämäläinen et al. [32], which proposes a sequential Monte Carlo approach for online motion synthesis. In their work sequential Monte Carlo sampling generates complex get up strategies and balancing behaviors without the use of precomputation or training data.

We do not aim to explain every detail of their implementation. Instead, we explain some basic concepts, starting with explaining how a fitness function can be adaptively estimated using sampling and a kd-tree. Then, we explain an extension of the kd-tree sampling to sequential estimation. Last, we explain how the sequential sampling method can generate optimal control signals in the motion synthesis problem.

3.3.1 Adaptive importance sampling using a kd-tree

The particular variant of SMC used is based on mutated kd-tree importance sampling [31], in which the evaluated samples are used to form a kd-tree which estimates the function, rather than using the samples as is, as in conventional SMC.

Figure 3.2 shows a one-dimensional example of how a kd-tree subdividing the parameter space can be used to construct a piecewise constant approximation of the fitness function $f(\hat{\mathbf{y}})$. The tree leaves i subdivide the parameter space into hypercubes of volume V_i.

Using the kd-tree, sampling can be based on importance, i.e. focused in areas of the parameter space with high expected fitness. Samples are generated by first randomly choosing a hypercube i with a selection probability proportional to

$$w_i = f(\hat{\mathbf{y}}_i) V_i. \tag{3.2}$$

Note that in contrast to Equation 3.1, w_i does not depend on $q(\hat{\mathbf{y}})$, which allows the insertion of samples from external sources where $q(\hat{\mathbf{y}})$ is not known.

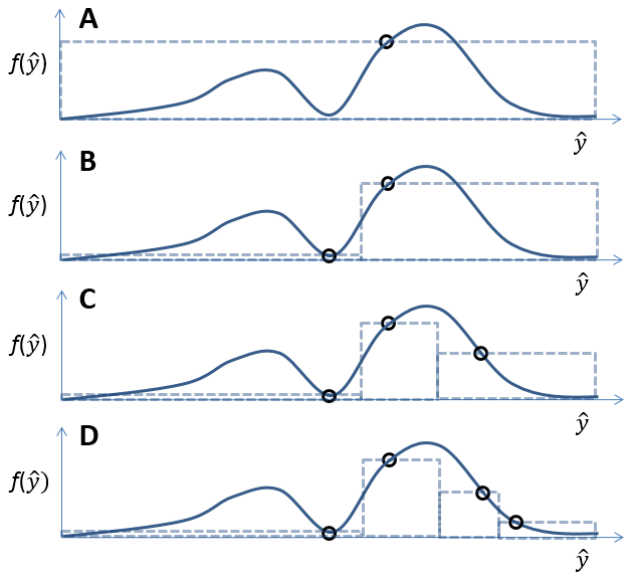


Figure 3.2: Adaptive importance sampling using a kd-tree. The evaluated samples defining the tree are shown as circles, and the kd-tree leaf nodes are shown as dashed lines.

A sample is then drawn from the proposal distribution, which is a multivariate Gaussian distribution $\mathcal{N}(\hat{\mathbf{y}}_i, \mathbf{C}_i)$. The covariance $\mathbf{C}_i$ is diagonal with elements:

$$c_{ij} = (\sigma d_{ij})^2, \tag{3.3}$$

where σ is a scaling parameter and d_{ij} is the width of the hypercube along dimension j. The Gaussian is used for sampling to introduce "blurring", which improves the sampling in cases where an unluckily chosen sample is not representative of the fitness inside the entire hypercube.

Each evaluated sample is added into the tree, subdividing the tree further at each step, and thus refining the sampling prior for the next samples. Similarly to Bayesian optimization, each evaluated sample immediately affects the generation of new samples, rather than just affecting the next generation as in CMA-ES, or the next time step as in particle filtering.

Although the adaptive sampling method is in principle sequential, samples can be evaluated in parallel, while synchronizing only the tree manipulations. In the context of trajectory optimization, the cost of the tree manipulations is insignificant in relation to the sample evaluation cost, i.e. evaluating the effect of the control signal using physics simulation.

3.3.2 Sequential kd-tree sampling

Next, we describe how the time-invariant adaptive kd-tree sampling can be extended for a time-varying fitness function.

The sequential sampling method is presented as lengthy pseudocode in Algorithm 1. We repeat the whole algorithm here for completeness, but for conciseness, we skip explaining some details of the algorithm, such as the tree manipulations.

The latter half of the algorithm (lines 17-29) is on a general level familiar: it draws N random samples using adaptive kd-tree sampling as in the time-invariant case. In the first half (lines 2-16) the kd-tree used in the previous time step is transformed for use in the next time step.

First, the tree is pruned to M samples (lines 2-5) from the original N. Then, the M samples are reinserted to the tree in a random order (lines 6-10) to avoid bias caused by the order in which the tree was constructed.

The next section (lines 11-15) inserts samples into the kd-tree based on heuristics and machine learning predictions. Inserting the evaluated samples into the kd-tree can be seen as constructing an implicit prior for the sample generation through the density estimation performed by the kd-tree. We will defer further discussion about machine learning to Chapter 6, where we discuss how to generate the predictions, and introduce alternative ways to incorporate machine learning into the sampling framework.

In addition to adaptive sampling of the whole parameter space using the kd-tree, the algorithm also includes a greedy local search, which is not explained in the pseudocode. For the last N_g samples, only the node corresponding to the sample with highest fitness is used for sampling, i.e. the random selection in line 18 is replaced. A smaller scaling parameter σ_g is used when sampling from the proposal Gaussian.

3.3.3 Parametrization

To use the adaptive optimization algorithm for motion synthesis, the control signals need to be represented as vectors $\hat{\mathbf{y}}$ in some real space $\mathbb{R}^n$. The chosen representation is a cubic spline, which defines time-varying joint angles that are used as target angles for the character's actuators. Additionally, the spline defines time-varying limits for the maximum torques applied by the actuators.

The spline is represented as a vector consisting of components for the spline's N control points:

$$\hat{\mathbf{y}} = [\hat{\mathbf{y}}_1^T, ..., \hat{\mathbf{y}}_N^T]^T, \tag{3.4}$$

Algorithm 1 kD-Tree Sequential Importance Sampling

for each time step t_j **do**
  // Prune tree to M samples
  while #samples $> M$ **do**
    find leaf i with minimum w_i
    RemoveTree($\hat{\mathbf{y}}_i$)
  end while
  // Randomly shuffle and rebuild tree using old fitnesses
  ClearTree()
  $\{\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_M\} \leftarrow$ RandomPermute($\{\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_M\}$)
  for $i = 1 \ldots M$ **do**
    InsertTree($\hat{\mathbf{y}}_i$)
  end for
  // Draw guesses from heuristics and ML predictors
  for $i = 1 \ldots K$ **do**
    $\hat{\mathbf{y}}_g \leftarrow$ DrawGuess()
    evaluate $f(\hat{\mathbf{y}}_g; t_j)$
    InsertTree($\hat{\mathbf{y}}_g$)
  end for
  $\{w_1, \ldots, w_{M+K}\} \leftarrow$ UpdateLeafWeights()
  // Then, perform adaptive sampling
  repeat
    Randomly select leaf node i with probability $\propto w_i$
    if node contains old fitness $f(\hat{\mathbf{y}}_i; t_{j-1})$ **then**
      compute current fitness $f(\hat{\mathbf{y}}_i; t_j)$
      $w_i \leftarrow V_i f(\hat{\mathbf{y}}_i; t_j)$ ▷ update weight
    else
      draw a sample $\hat{\mathbf{y}}_{new} \sim \mathcal{N}(\hat{\mathbf{y}}_i, \mathbf{C}_i)$
      Evaluate $f(\hat{\mathbf{y}}_{new}; t_j)$
      $\{n_1, n_2\} \leftarrow$ InsertTree($\hat{\mathbf{y}}_{new}$)
      $w_{n_1} \leftarrow V_{n_1} f(\hat{\mathbf{y}}_{new}; t_j)$
      $w_{n_2} \leftarrow V_{n_2} f(\hat{\mathbf{y}}_{n_2}; t_j)$ ▷ $f(\hat{\mathbf{y}}_{n_2}; t_j)$ known
    end if
  until #samples $= N$
end for

where each control point is defined as

$$\hat{\mathbf{y}}_i = [\mathbf{q}_i^T, \mathbf{l}_i^T, t_i]^T. \tag{3.5}$$

The vector $\mathbf{q}_i$ specifies the joint angles for the whole character. The effect of this parametrization is that the character must time the movements of the degrees of freedom synchronously, rather than controlling each degree of freedom completely independently.

The vector $\mathbf{l}_i$ specifies the maximum torques, but instead of using a separate number for each actuator, the torques are specified for only three groups of actuators: torso, arms, and legs.

The scalar t_i specifies the length of the spline segment, i.e. the spline is non-uniform and the length of each segment is included as an optimizable parameter. The scalars represent relative offsets from the current time, rather than absolute points in time.

For use in the kd-tree sampling scheme, the optimized variables need to be defined for finite ranges. The minimum and maximum for angles $\mathbf{q}_i$ are naturally defined by the physical range limits of the joints. The minimum and maximum for torques $\mathbf{l}_i$ are set to some values suitable for the specific application. The range of the segment length t_i is limited by a manually adjusted t_{min} and by $t_{max} = t_{horizon}/N$, where $t_{horizon}$ is the longest allowed total length of the control signal.

To evaluate the fitness of each $\hat{\mathbf{y}}$, the physics of the character are simulated forward in time using the continuous control signal defined by the spline. The actual simulation method is largely irrelevant, that is, the simulation is treated as a black box. The realized motion is sampled at time intervals Δt up to $t_{horizon}$ to form a representation $\mathbf{S}$, which is used by the fitness function defined as $f(\mathbf{S})$.

As mentioned previously, the sequential sampling method allows the insertion of heuristic guesses. In previous work and in the system implemented for this thesis, the best sample of the previous frame is added to the optimizer after stepping the control signal forward by Δt.

3.3.4 Fitness function

We present two fitness functions in this thesis, one for the locomotion test case in Chapter 4 and one for the reference animation tracking in Chapter 5. A third example of a fitness function can be found in the original paper by Hämäläinen et al., where the fitness function weighs balancing and get-up behaviors.

The $f(\mathbf{S})$ used to evaluate the samples can be any function that maps to non-negative real numbers, though the two fitness functions presented in

this thesis share a common form, which is very similar to the form used by Hämäläinen et al.:

$$f(\mathbf{S}) = e^{-\frac{1}{2}E_{average}} e^{-\frac{1}{2}E_{terminal}} e^{-\frac{1}{2}E_{smoothness}}. \tag{3.6}$$

The components of the function, which are explained below, are functions of $\mathbf{S}$, but we omit this in the notation for brevity.

The average cost

$$E_{average} = \frac{1}{N}\sum_{i=1}^{N} ||\mathbf{d}_{average}(i)||^2, \tag{3.7}$$

measures the deviations $\mathbf{d}_{average}$ from some desirable state to the realized motion for the N sampled frames in $\mathbf{S}$.

The terminal cost, which ignores the effect of the first T frames, is modeled as

$$E_{terminal} = \min_{T<i\leq N} ||\mathbf{d}_{terminal}(i)||^2, \tag{3.8}$$

for some deviation measure $\mathbf{d}_{terminal}$. The cost is measured as the minimum over a few frames, allowing some slack in the timing to reach the target state and thus smoothing the optimization landscape.

Finally, the third factor in the total fitness function is the smoothness cost

$$E_{smoothness} = \frac{\mu_a}{\sigma_a^2} + \frac{\mu_J}{\sigma_J^2}, \tag{3.9}$$

which penalizes high mean squared accelerations μ_a and mean squared jerks (time-derivatives of acceleration) μ_J of the character's bones. Our scaling factors are $\sigma_a = 10$ and $\sigma_J = 20$, which are slightly less strict than the values in the original paper.

Chapter 4

Data-driven sequential Monte Carlo motion synthesis

In the next three chapters, we describe the main contribution of this thesis: the data-driven sequential Monte Carlo motion synthesis framework. In this chapter, we start off with a broad overview of the different components of the implemented system. We also present the locomotion test case, which shows how the system works in practice.

4.1 System overview

The main goal of the system is to extend the sequential Monte Carlo motion synthesis framework by using prior knowledge to direct the optimization process. We would like the system to follow reference animation data where possible, but we would also like the system to be able to generate good solutions for situations where reference data is not available.

Figure 4.1 shows a coarse overview of the components of the system. The central piece is the online optimization component, which is the sequential Monte Carlo optimizer described in the previous chapter.

As before, the optimal control signal drives the simulated actuators, which in turn generate torques that move the simulated rigid bodies. Section 4.2 describes the physical model in more detail.

The fitness function for the optimizer is composed using the user's input and optionally other application-specific logic. We give a concrete example of a fitness function in Section 4.4, where we explain the locomotion test case.

The system is made data-driven by the reference animation tracking component and the associated machine learning component. Reference animation data authored by an animator is fed into the tracking component, which uses

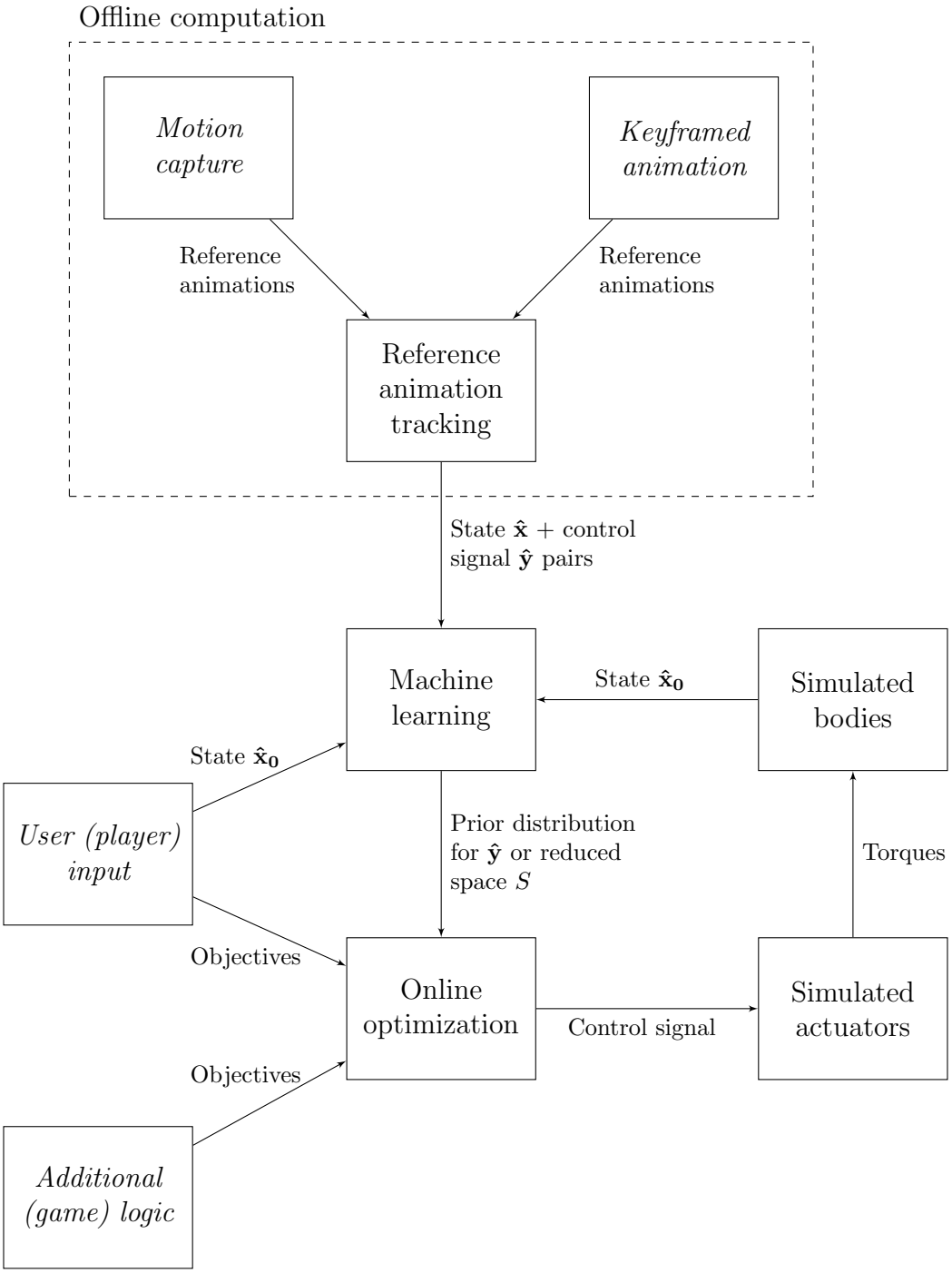


Figure 4.1: System overview chart. Rectangles represent parts of the system: *italic type* for external parts, normal type for implemented parts. Arrows represent moving data. The dashed line seperates the offline component.

the physical character to reproduce the kinematic reference data. The tracking also uses sequential Monte Carlo optimization, though this is not made explicit in the overview chart. The tracking component is discussed in detail in Chapter 5.

The purpose of the tracking component is to generate examples of control signals that may be used to direct the online optimization process. For example, in the locomotion case running animations are tracked to generate examples of control signals that generate a running motion.

On the chart, these examples are shown as $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ pairs. The variable $\hat{\mathbf{y}}$ is a control signal that optimally follows the reference animation, while variable $\hat{\mathbf{x}}$ represents the physical state of the character for which the optimal control signal was generated. The specific form of $\hat{\mathbf{x}}$ is explained in Chapter 6.

The machine learning component finds the best estimate for control signal $\hat{\mathbf{y}}$, given a state $\hat{\mathbf{x}}_\mathbf{0}$, which may not be found in the set of examples. The estimate is represented as a prior distribution or as a reduced space. Chapter 6 has details.

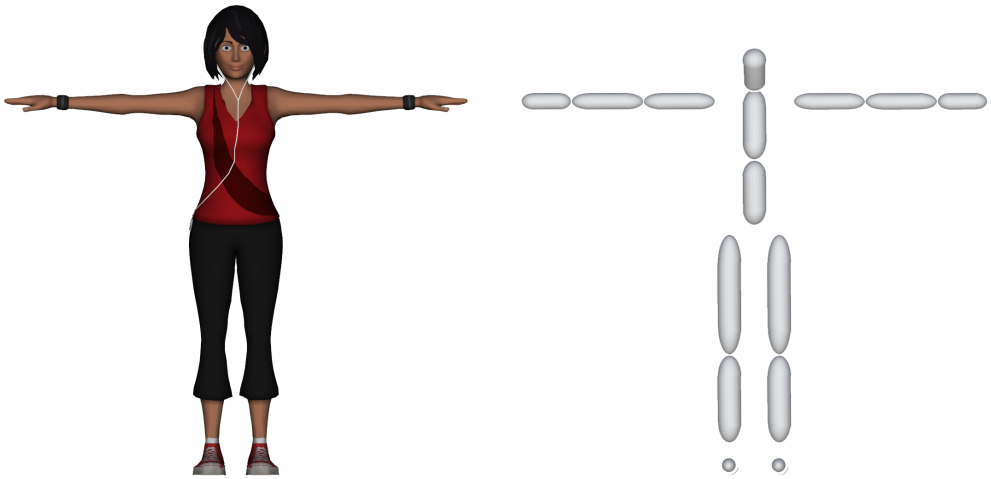
The previous work by Hämäläinen et al. [32] also included a machine learning component, though it was used for a subtly different purpose. We use the component to learn how to generate motion similar to the reference animations, while the previous work used the component as a type of cache to remember the previous solutions found by the optimizer.

4.2 Simulated physical character

As the physics model and the implementation are for the most part as in the previous work by Hämäläinen et al. [32], we will skip most details, and focus on the differences to the previous system.

The physical character consists of 15 capsule-shaped rigid bodies of a constant density. The physics configuration can be seen in Figure 4.2 together with the visual representation. We also experimented with box-shaped rigid bodies for the feet and found that the quality of locomotion tasks was in general improved. However, we selected capsules for the interactive locomotion test case, as collision detection of capsules is more stable when using large time steps. Self-collisions are not enforced: the rigid bodies of the character collide with the ground and other objects in the scene, but not with the character's other rigid bodies.

The rigid bodies are for the most part connected using joints with three degrees of freedom (3-DOF). For elbows and knees, we use 1-DOF joints. When the unactuated 6 degrees of freedom for the root are included, the total amount of degrees of freedom for the whole character is 40. One dif-



(a) Visual representation: a skinned mesh

(b) Physical representation: rigid bodies corresponding to major body parts

Figure 4.2: The two representations of the simulated virtual human.

ference to previous work by Hämäläinen et al. is the ankle joint, which was changed from 1-DOF to 3-DOF to allow some agile locomotion movements. Additionally, joint limits were made wider than in previous work.

The joints have associated motors which provide actuation for the character. The motors apply the limit $\mathbf{l}$ on the maximum applied torque received from the optimizer. The motors attempt to match a target velocity, which is computed using the difference between the current joint angle and the target joint angle $\mathbf{q}$ given by the optimizer.

The basic skeletal animation is handled by Unity game engine's Mecanim animation system. Instead of using Unity's default physics engine PhysX, the Open Dynamics Engine (ODE) physics engine is integrated into Unity, since the default engine does not allow simulating multiple physics states in separate threads decoupled from the game state.

The constraint handling LCP problem is solved using the pivoting algorithm implemented in ODE, which is based on Danzig's algorithm [14]. We experimented with the alternative iterative solver in ODE, but found that the pivoting algorithm was a better choice for simulating the complex character, especially when using large simulation time steps.

The constraint force mixing (CFM) feature in ODE was enabled to soften the contacts between the character and the ground. In addition to making the LCP problem easier to solve, the softness also was found to slightly improve the robustness of the optimization in some locomotion tasks. Some research

suggests that soft physics simulation in general helps with the robustness and quality of character control [36].

To allow real-time performance, the simulation frequency was set to 30 Hz. The low frequency caused collisions to be detected inaccurately, which in turn caused additional jitter through foot contacts in locomotion tasks.

4.3 Preprocessing reference animations

The reference animations used by the system are created by an animator using standard computer animation content creation tools. The data may originate for example from motion capture or from manual keyframing.

The reference animations are treated as a continuous signal of body positions and orientations. The continuous signal is sampled at discrete intervals using the simulation time step Δt. This allows for easy frame-by-frame comparisons in the fitness function. We sample the absolute positions and orientations for each body part, as well as the relative angles for each joint degree-of-freedom.

In addition to the kinematic data sampled from the reference animation, some dynamics data is required for the fitness function. As the reference animations only include the positions and orientations of the body parts, we need to infer the velocities. For this, we use the finite differences approximation. The linear velocities are easy to approximate:

$$\mathbf{v}_t \approx \frac{\mathbf{p}_{t+\Delta t} - \mathbf{p}_t}{\Delta t} \tag{4.1}$$

Approximating the angular velocity (bi)vector using the orientation quaternions is a little less obvious. First, we define a unit quaternion representing the difference between subsequent time steps

$$\mathbf{q}_{diff} = \mathbf{q}_{t+\Delta t}\mathbf{q}_t^{-1}, \tag{4.2}$$

then, we use its axis-angle representation

$$\mathbf{q}_{diff} = \cos(\frac{\theta}{2}) + \sin(\frac{\theta}{2})(in_x + jn_y + kn_z) \tag{4.3}$$

to find the angular velocity:

$$\boldsymbol{\omega}_t \approx \frac{\theta \mathbf{n}}{\Delta t}. \tag{4.4}$$

We also compute the position and linear velocity of the center of mass. These two are simply the averages of the positions and linear velocities of each body part weighted by the mass distribution of the simulated character.

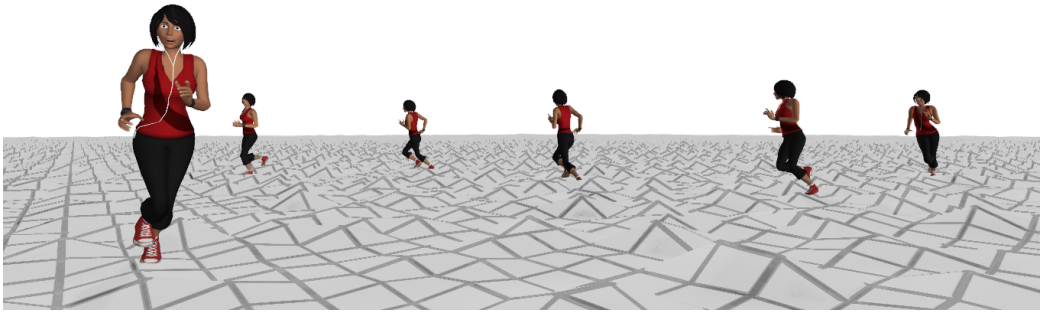


Figure 4.3: Locomotion test case: character navigating through bumpy terrain.

4.4 Locomotion test case

We test the implemented system in practice with a simple real-time locomotion test case. Figure 4.3 shows an example, in which the character runs through uneven terrain at a constant speed, while the user of the system interactively chooses target directions for the character.

To define the running motion, the system is fed with example kinematic running motions. For the simplest version of the test case, we use a single motion, in which the character runs along a straight line. Other motions, such as starting, stopping, and turning, could also be used.

The fitness function is split into average and terminal fitnesses, as described in Chapter 3. The deviation vector for the average cost consists of four components, which we will describe next:

$$\mathbf{d}_{average}(i) = \left[\frac{\mathbf{a}(i)}{\sigma_{a1}}^T, \quad \frac{\mathbf{v}(i)}{\sigma_{v1}}^T, \quad \frac{\mathbf{r}(i)}{\sigma_{root}}^T, \quad \frac{\mathbf{c}(i)}{\sigma_{bvel}}^T \right]^T. \tag{4.5}$$

The scaling constants σ are hand-tuned to define the acceptable deviation for each component and to weight their relative importances.

The first three components aim to keep the synthesized motion close to the animations in the training set. The effect of the components is small during normal running, but becomes significant when the character is faced with external disturbances, such as pushes.

The $\mathbf{a}(i)$ and $\mathbf{v}(i)$ components measure the joint angle and joint angle velocity differences of simulated frame i to animations in the training data set. Instead of comparing the difference to a single frame j in the training data set, the measures compare the difference to all j and use only the frame with the smallest difference in the final measure.

The root bone orientation component $\mathbf{r}(i)$ works similarly. Instead of inspecting the global orientation quaternions, we use a local descriptor that is invariant to rotations along the y axis. The descriptor is used to allow, for example, using a single straight running motion in the training set to synthesize running in every horizontal direction. The local descriptor is created by transforming the x, y, and z unit vectors defined in the root bone's local space to the global space, and using the three y components of the transformed vectors.

The fourth component, $\mathbf{c}(i)$, measures the vector difference between the current velocity of the center of mass to the target velocity. In the simplest locomotion test case with only one running animation the function is non-trivial: instead of letting the user of the system directly control the target velocity, we use a separate constrained velocity vector in the fitness function. The vector is updated each frame towards the user's target velocity, but only up to a maximum angular velocity, and only if the best fitness of the previous frame is over a certain threshold. This constraint keeps the character from attempting tight turns and falling.

The target velocity component is the only aspect in the fitness function specific to locomotion. In principle, the fitness function should be usable for other applications just by replacing the velocity component.

The total fitness function also includes the terminal cost, which has only two components:

$$\mathbf{d}_{terminal}(i) = \left[\frac{\mathbf{a}(i)}{\sigma_{a2}}^T, \quad \frac{\mathbf{r}(i)}{\sigma_{root}}^T \right]^T. \tag{4.6}$$

The function $\mathbf{a}(i)$ and $\mathbf{r}(i)$ are just as in the average cost. The terminal cost does not primarily affect the motion quality as the average cost does, instead, the terminal cost helps with keeping the character balanced.

The locomotion test case uses a relatively short time horizon with maximum length $t_{horizon} = 1$ s. This is due to the minimum spline segment length of $t_{min} = 0.1$ s, which is short to allow the quick running motions to be accurately replicated, and the number of segments $N = 7$, which is low to decrease the number of optimization parameters. Despite the relatively short horizon, the total number of optimized parameters is 266, which is considerably higher than the 136 parameters in the original balancing case by Hämäläinen et al.

To improve the motion quality, we manually override the orientation and position of the head each frame as a post-processing step. The orientation of the head is set to point towards the constrained velocity target and the position is set to keep the neck length consistent. In addition to smoothing the movement of the head, the override also helps to make the character seem more human, as she appears to anticipate the turns.

4.5 Summary

The original sequential Monte Carlo motion synthesis paper presented a test case in which a character could balance and synthesize complex getting up motions without using data to drive the optimization process. The locomotion test case presented here serves as a good test case for the data-driven system, as synthesizing natural-looking locomotion using only the original unguided optimization is not feasible using a practical amount of samples.

In the described framework the low level details of synthesized motions are described by the reference animations, which allows - in theory - an animator to easily control the resulting motions. Constructing a good fitness function is a tedious process of trial and error, but generating reference animations is relatively fast and straightforward, especially when using motion capture.

The fitness function presented for the locomotion test case favors generality and simplicity over efficiency: it describes the motion only at a high level by using the target velocity. Although we use a running motion, in principle, any motion that moves the character forward should work.

We will return to the locomotion test case in Chapter 7, where we present results of the system in action.

Chapter 5

Tracking reference animations

In the previous chapter, we gave an overview of the complete data-driven sequential Monte Carlo motion synthesis system. In this chapter, we explain the implementation of the offline component of the system, which transforms kinematic reference animations to data usable by the online component. This is performed by tracking the reference animations with the simulated character.

So far, we have only presented the reference animation tracking component as a helper for the online optimization, but there are also other potential uses for the component. For example, as the component synthesizes physically-correct motion out of kinematic data, it could be useful as a post-processing step for kinematic motion synthesis.

We start off by giving a general idea on how to apply the sequential Monte Carlo optimization algorithm to tracking. Then, we present the details: the fitness function and the sample generation.

5.1 Overview

We repeat the relevant part of the system overview chart in Figure 5.1. As seen on the chart, the goal is to compute example state $\hat{\mathbf{x}}$ and control signal $\hat{\mathbf{y}}$ pairs for the use of the online component. To compute the pairs, we combine the state of each frame of the reference animation with the corresponding optimal control signals.

Since the tracking result is computed offline, we emphasize motion quality and robustness over performance. Our goal is to track all kinds of reference motions as closely as possible while maintaining physical correctness. We should keep the performance good enough so that the user can quickly review the tracking results, but we do not aim for real-time performance. For quicker

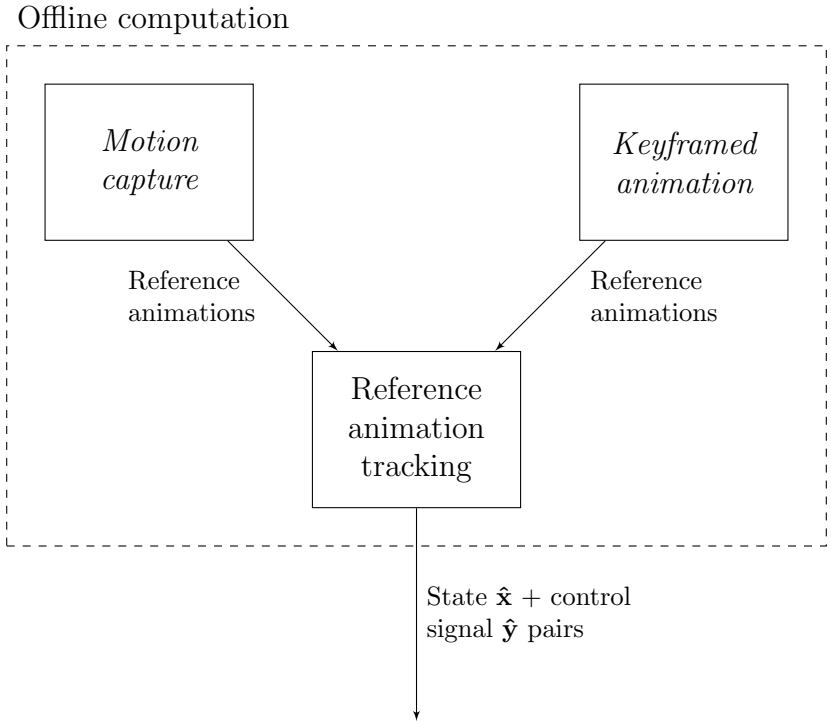


Figure 5.1: The offline computation part of the system overview chart.

previews, it is also possible to temporarily lower the amount of samples generated.

As we cannot simulate the actual human performer of the motion data with complete accuracy, the tracking system should be robust enough to allow for differences in body proportions, masses, joint range limits, etc.

Next, we explain the tracking algorithm and the data extracted from the reference animations.

5.1.1 Optimization

The optimization is performed as follows:

1. Set $t = 0$. Initialize by posing the simulated character using the positions and the velocities of the first frame in the reference animation

2. Find optimal control signal $\hat{\mathbf{y}}$ using sequential Monte Carlo, where fitness is defined as minimizing the deviation from the reference animation

from time t forward. Store current state $\hat{\mathbf{x}}$ and optimal control signal $\hat{\mathbf{y}}$.

3. Apply the optimal control signal to character's actuators and step the physics state forward.

4. Set $t = t + \Delta t$. Return to step 2 if $t < T$, where T is the length of the reference animation.

In some cases, we may choose to also include the initial state as an optimization variable, for example if the first frame of the animation violates the joint range limits.

5.1.2 Reference data

The optimization process is independent of the specific content creation tools, but in general, motion-captured animations are close to being physically correct and are thus easier to track with the simulated character.

In some applications, characters are animated to perform motions that could not be performed by a human in a motion capture setting. For example, characters in video games often jump without first performing the windup motion required by humans. If the reference motion is in this way entirely physically impossible, the optimization algorithm may have trouble finding a solution.

It would be possible to extend the framework for these types of motions by adding a special force as an optimization parameter. This special force would provide extra momentum if the character's actuators are found insufficient. The use of the special force could be penalized in the fitness function to keep the motion primarily unassisted.

Note that the reference animation may only be successfully tracked if the simulated environment roughly matches the environment in which the motion was originally performed. That is, the scene's geometry, friction, and restitution should be modeled to some degree. Fortunately, for most motions, a simple ground plane of average friction and restitution is accurate enough.

In our implementation, the reference animations are retargeted onto the target character by the Unity game engine's Mecanim animation system. The retargeting algorithm used does not try to preserve the physical correctness of the motion. In theory, the retargeting step is optional and it should not affect the optimization in a major way. However, we did not attempt to prove this, as Mecanim does not allow one to disable the retargeting without also disabling other parts of the animation system.

5.2 Fitness function

Two criteria are used in modeling the fitness function. The obvious criterion is to keep the simulated character close to the reference motion at all times. The secondary and less obvious criterion is to help to direct the optimization process towards regions of high fitness. It is not sufficient to provide a fitness function that can weigh a perfect optimization result correctly - the function must also be smooth enough so that the space near the perfect result can direct the search.

Our fitness function is based on the work of Liu et al. [48] and is adapted to fit the sequential Monte Carlo framework. The total fitness function is given as $f(\mathbf{R}, \mathbf{S})$, i.e. it is a function of the reference motion $\mathbf{R}$ and the simulated motion $\mathbf{S}$.

The fitness function is a product of average, terminal, and smoothness factors as described in Chapter 3. In this case, the average cost defines the quality of the motion, while the terminal cost maintains balance and ensures that the simulated character will end up in a state from which the motion can be continued.

5.2.1 Average cost

The frame deviation vector used for the average cost is a concatenation of the joint angle, body angular velocity, and head orientation components:

$$\mathbf{d}_{average}(i) = \left[\frac{\mathbf{a}(i)}{\sigma_{a1}}^T, \quad \frac{\mathbf{v}(i)}{\sigma_{v1}}^T, \quad \frac{\mathbf{h}(i)}{\sigma_{head}}^T \right]^T \tag{5.1}$$

The joint angle component $\mathbf{a}(i)$ is simply the vector difference in radians between the reference joint angles and the simulated joint angles in frame i.

Similarly, the body angular velocity component $\mathbf{v}(i)$ is the difference of the angular velocities of each rigid body, measured in radians per second.

Lastly, the $\mathbf{h}(i)$ component stabilizes the perceptually important movement of the head. The orientation quaternions of the reference motion and the simulated motion are compared for both the head and the chest. Chest orientation deviation is penalized since the orientation of the chest has a large effect on the position and orientation of the head. The quaternions are compared using the function:

$$\theta(\mathbf{q}, \mathbf{r}) = 2 \arccos(\mathbf{q} \cdot \mathbf{r}) \tag{5.2}$$

Without the head stabilization component, the head jitters in an unnatural way. This extra emphasis on the head is also biomechanically justified, as humans have the distinct ability to keep their head stable even when performing high-energy movements, such as running.

5.2.2 Terminal cost

The deviation vector used for the terminal cost is similar to the average cost vector, but a little longer:

$$\mathbf{d}_{terminal}(i) = \left[\frac{\mathbf{a}(i)}{\sigma_{a2}}^T, \quad \frac{\mathbf{v}(i)}{\sigma_{v2}}^T, \quad \frac{\mathbf{r}(i)}{\sigma_{root}}^T, \quad \frac{\mathbf{s}(i)}{\sigma_{rvel}}^T, \quad \frac{\mathbf{e}(i)}{\sigma_{end}}^T, \quad \frac{\mathbf{b}(i)}{\sigma_{bal}}^T, \quad \frac{\mathbf{c}(i)}{\sigma_{bvel}}^T \right]^T \tag{5.3}$$

The first two components, $\mathbf{a}(i)$ and $\mathbf{v}(i)$, are as described previously for the average cost.

The next two, $\mathbf{r}(i)$ and $\mathbf{s}(i)$, represent the deviation of the root orientation and the root angular velocity. As before, Equation (5.2) measures the orientation difference and the angular velocity component is again the difference in radians per second. The root is emphasized in this way because of the difficulty of returning to the reference motion after a large root displacement.

Next, $\mathbf{e}(i)$ measures the deviation of the end effector heights. For the hands, two heights are measured: one for both hands. For the feet, we measure two points on both feet: the heel and the ball of the foot. Measuring these two points rather than just a single point makes some agile locomotion animations easier to track. In general, the end effector component is the most important component for locomotion tasks.

For measuring balance, we add $\mathbf{b}(i)$. For this, we need to define a horizontal vector comparing the positions of the center of mass (CoM) and end effector j. Here we use a total of four end effectors: two feet and two hands. The vector is:

$$\mathbf{m}_j = (\mathbf{p}_{CoM} - \mathbf{p}_j)|_{y=0} \tag{5.4}$$

The differences of each end effector are combined to a single vector:

$$\mathbf{m} = \left[\mathbf{m}_1^T, \quad \ldots, \quad \mathbf{m}_M^T \right]^T \tag{5.5}$$

The balance measure $\mathbf{b}(i)$ is then simply the difference in meters between the reference and the simulated motion. Liu et al. [48] provide a detailed justification for this measure. We note only that the measure is proved empirically effective and is found necessary for keeping the character in balance.

Finally, the component $\mathbf{c}(i)$ measures the difference between the linear CoM velocities of the reference and simulated motions in meters per second. This causes the character to move in the environment instead of trying to perform the motions in place.

5.2.3 Analysis

It is important to notice that the total fitness function is independent of the absolute horizontal position of the character. Adding this dependence

to the fitness function would be trivial, but it would limit the ability to generalize the motion to different types of characters. It would also make the optimization algorithm more sensitive to mistakes in the beginning or in the middle of the motion, as it is difficult to shorten the distance between absolute positions after a mistake has happened.

Additionally, we note that only a few components of the fitness function are specific to humans and none of the components are specific to a certain type of motion. Therefore, the fitness function should be applicable to a wide range of uses.

The total fitness function contains a high number of hand-tunable constants σ: a total of 10, if ignoring the smoothness factor. Fortunately, most of the constants are easy to roughly adjust to the correct value, as they correspond to intuitive features of the simulated motion. Even more importantly, the quality and the performance of the tracking was empirically found to not depend on the exact values of these constants. As an example, it is easy to reason what the allowed deviation in the end effector heights should approximately be: our value is 10 cm, but 5 cm or 20 cm would work just as well. Also, the constants need not be tweaked separately for different types of motion.

5.3 Sample generation

The previous section explained how the samples are weighed, but how do we choose which samples to weigh? The naive approach would be to generate the samples using only the adaptive importance sampling mechanism to guide the search. However, sampling in this way has an extremely high change of failure: the sampling space is large and high-dimensional, whereas the areas of high fitness are tiny.

We can make the optimization algorithm work by using two observations. The first observation: at each frame, we know that trying to follow the exact reference joint angles would at least result in a sample that is close to the optimal in the sampling space. All optimal samples have this feature, since high deviation from the reference joint angles would result in a high cost in the joint angle difference component of the fitness function. Figure 5.2 shows an example of typical reference and optimal signals.

The second observation is as follows: if we have an optimal result for the previous frame, we can wind the previous result forward by one step and get a new sample that covers the majority of the time horizon.

To apply the first observation to our advantage, we need to somehow fit the reference motion to the control signal's parametrization. The reference

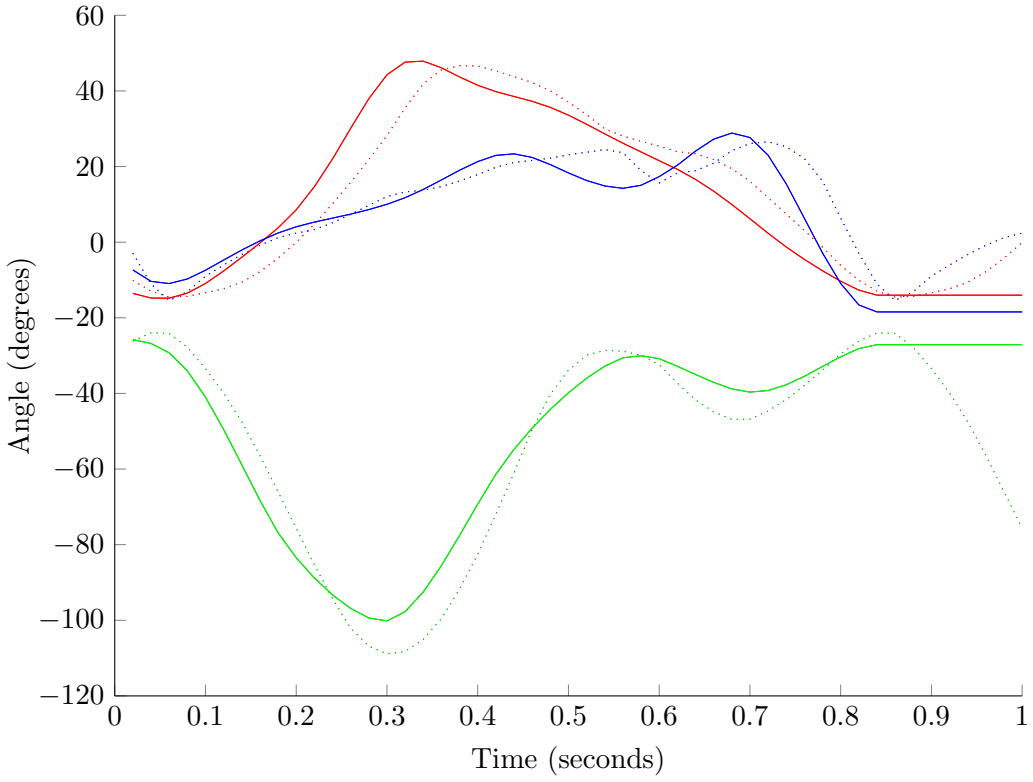


Figure 5.2: The optimized control signal (—) and the reference motion (- - -) of a run cycle animation. The colored curves represent the angles along the sagittal axis of the character's hip, knee, and ankle. The optimized control signal flattens out after about 0.8 s because of the limit on the spline's length. Note how the reference ankle motion is detailed and discontinuous because of contacts, but the control signal is smooth. Note also the misalignment of some of the peaks of the two functions.

motion may be generated by any function, but our parametrization is a spline. We might, for example, find the least-squares fit of the spline to the reference function. However, the resulting spline would only be optimal in the least-squares sense and may not be optimal as a control signal. Additionally, we should note that we are not limited to finding a single best fit: we may generated multiple samples.

For these reasons, we generate a number of random fittings to the reference function and let the fitness function evaluate their optimality. The spline segment durations are picked uniformly at random in the range defined by the framework. The target joint angles are then picked from the reference

animation corresponding to those points in time.

The control signal parametrization also includes the maximum forces applied by joint motors. Unfortunately, the reference animation contains only kinematic data, and there is no way to infer the actual forces applied in the reference motion. The maximum forces have little effect on the tracking result, but they would be significant in the online component for tension behaviors and generalization under disturbances.

To exploit the second observation, we employ the usual sequential Monte Carlo sample generation heuristics used by Hämäläinen et al [32]. The best sample from the previous frame is kept and evaluated in the next frame. As usual, old samples are wound forward in time to better match the new frame. Unlike in other applications of the sequential Monte Carlo framework, no samples are generated completely at random.

In addition to generating the random reference animation splines from scratch the way described above, we also use the reference animation data to extend the previous best sample. We simply add a new segment to the end of the previous best sample in the same way we generated the completely new random reference animation splines. The reason for doing this requires an explanation: if there is a particularly difficult segment within the time horizon, the reference animation splines generated from scratch might all fail. However, the optimization algorithm may already have generated a good solution to overcome the difficult segment, so we use the previous best solution for this segment instead of generating the whole spline from scratch.

The amount of samples generated in all the ways described previously are exposed as adjustable parameters in our implementation. By default, 25% of the samples are random reference animation splines, 5% are random extensions to the previous best, and 50% of the samples are generated greedily. In addition, 25% of the samples from the previous frame are kept to form the initial sampling distribution.

5.4 Summary

In this chapter, we have shown how the sequential Monte Carlo framework can be applied to track reference animations. We defer detailed analysis of the results to Chapter 7, but we summarize that the system does work: the system tracks various motions robustly, although with varying quality.

Our method of applying the sequential Monte Carlo framework to tracking is not the only possible one. We made a number of subjective choices, especially when constructing the fitness function. There is certainly room for improvement: for example, we measure motion similarity mostly by cal-

culating squared differences separately for each degree of freedom. This is not accurate as a measure of the perceptual difference [66], but it is simple to describe and implement.

Furthermore, the tracking component is not the only component affecting motion quality. Tracking can only work well if the simulated physical model is accurate. However, accurate simulation would require deeper inspection of the human biomechanics, which is beyond the scope of this thesis.

Chapter 6

Learning control from examples

The previous chapter explained how sequential Monte Carlo optimization can be applied to generate control signals $\hat{\mathbf{y}}$ using reference animations. In this chapter, our goal is to gather a set of $\hat{\mathbf{y}}$ examples computed offline to help the online optimization process. To reach this goal, we employ various machine learning methods.

In the overview section below, we explain the learning problem in detail. We also give some prerequisites on which the rest of the chapter is based on. In each of the sections following the overview we apply a specific machine learning method to the problem: approximate nearest neighbors (ANN) in 6.2, locally weighted regression (LWR) in 6.3, mixture of regressors (MoR) in 6.4, and finally self-organizing map (SOM) in 6.5.

6.1 Overview

Since the machine learning component works in tandem with the online optimization component, there is no need to learn how to control the character purely based on examples. Instead, our goal is to learn from the data something that helps to direct the optimization towards useful parts of the sampling space.

We repeat the relevant part of the system overview chart in Figure 6.1. In addition to the example control signals $\hat{\mathbf{y}}$, the offline component gives us $\hat{\mathbf{x}}$, which encode information about the state of the simulation that generated the corresponding control signals. In the online component, we are given $\hat{\mathbf{x}}_\mathbf{0}$, which encodes information about the current state. In Subsection 6.1.1 we will explain exactly what $\hat{\mathbf{x}}$ (and $\hat{\mathbf{x}}_\mathbf{0}$) contains.

We pair the two components together to form training samples $\hat{\mathbf{z}} = (\hat{\mathbf{x}}, \hat{\mathbf{y}})$. The training set, which contains all the training samples $\hat{\mathbf{z}}$, is named $\hat{\mathbf{Z}} =$

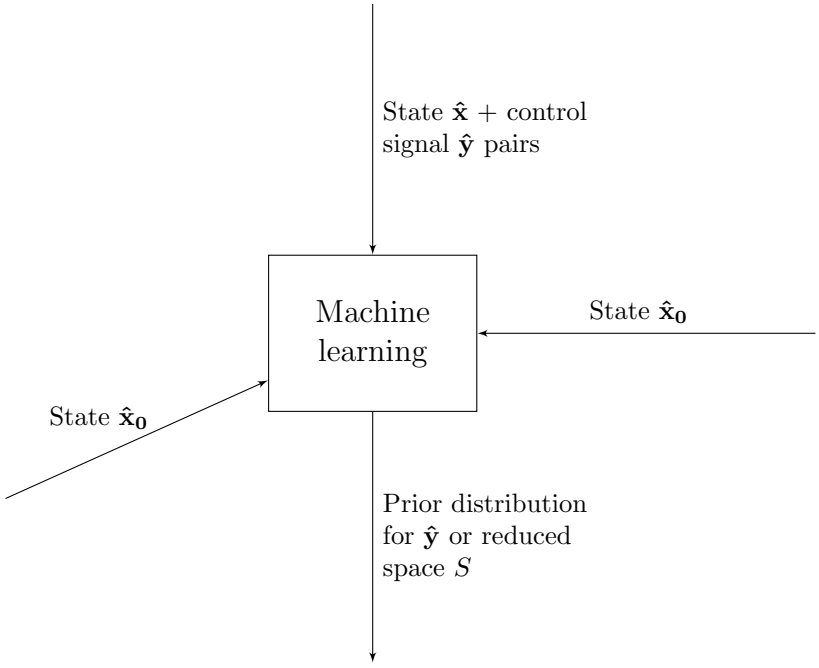


Figure 6.1: The role of the machine learning component in the system overview chart.

$(\hat{\mathbf{X}}, \hat{\mathbf{Y}})$. The training samples are normalized so that each $\hat{\mathbf{z}}$ component ranges from -1 to 1. Subsection 6.1.2 explains further preprocessing to the data.

The literature in machine learning describes a huge number of different methods others have successfully applied to their problems. Which criteria should we consider for choosing a learning method for our problem?

Obviously, we would like the method to model the relationship between $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ to allow for good generalization for $\hat{\mathbf{x}}_\mathbf{0}$ outside of the training set. The relationship in this case is nonlinear, discontinuous (because of contacts), and in general: complicated. We are interested primarily in supervised learning methods since our data consists of labeled $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ pairs. The $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ in question are high-dimensional, so the learning method should be suitable for highly multivariate problems.

The machine learning component is a part of a system used by animators and game designers, who are most likely not experts in machine learning. Therefore, the system should produce consistent results without requiring manual adjustments to the parameters.

The problem presented is a difficult one, but compared with many other machine learning problems, it does come with one convenient feature: we can generate any number of training samples ourselves. Although only a limited number of reference animations exist, we may simulate any sort of $\hat{\mathbf{x}}$ and optimize for $\hat{\mathbf{y}}$. Generating the samples is relatively quick: a few thousand samples can be generated in under an hour on a standard desktop computer.

To make use of this convenience, the machine learning method should be scalable to big training data sets in terms of computation and memory requirements. Though the speed of the training algorithm is not primarily what we are interested in, the method should not keep the user of the system waiting for hours.

For performance, the critical concern is the fast retrieval of $\hat{\mathbf{y}}$ when $\hat{\mathbf{x}}_\mathbf{0}$ is given. The goal is to direct the optimization process, but if the directions are not available quickly enough, it is better to spend the time on undirected optimization instead.

As the vector $\hat{\mathbf{x}}$ is only an approximation of the complete state of the simulated world, it cannot encode all the information needed to find out the optimal $\hat{\mathbf{y}}$. For example, we might include some information about the surrounding terrain in $\hat{\mathbf{x}}$, but including every fine feature of the terrain would be impractical. Thus, the machine learning method should be multimodal: it should be able to handle the ambiguity caused by the many-to-one mapping and consider multiple possible $\hat{\mathbf{y}}$ for a single $\hat{\mathbf{x}}$.

6.1.1 State features

In the previous chapters we have explained what the control signal $\hat{\mathbf{y}}$ consists of. In this section, we explore multiple possible components which may be used to construct the state feature vector $\hat{\mathbf{x}}$. In $\hat{\mathbf{x}}$, we may encode information about the character, the environment, and the goal.

A simple way to encode information about the state of the character in $\hat{\mathbf{x}}$ is to list the current joint angles. The joint angles have a convenient feature: they are invariant to changes in the global position and orientation of the character.

While invariance to the global pose is good in general, we would like to include some information about the orientation. For example, the joint angles alone do not disambiguate between standing on one's feet and standing on one's head. To allow for invariance for rotations along the y axis, we could use a local descriptor similar to the one used in the locomotion fitness function described in Chapter 4.

We might also want to list the heights of the character's body parts. The heights offer a more direct way to some important aspects of the current

state. For example, though it could be possible to infer through the leg's joint angles whether or not the foot is in contact with the ground, the information is more readily available as the height of the foot. The heights are retrieved from the simulation with downward ray intersection tests, encoding some information about the environment as well.

In addition to the positional features described above, we would like to list the velocities, since results of motions are very different when moving fast. Instead of encoding the velocity of each body part separately, we encode only the velocity of the center of mass. For invariance to orientations, instead of the encoding the velocity in global coordinates, we compute the velocity local to the root of the character.

In addition to the character and environment features, we may also wish to include information about the end result of the particular control signal, i.e. what applying the signal actually does to the character. For example, to disambiguate control signals for turning in place and walking forward, we could include features that encode the change in the character's velocity and heading. For the training samples, we can record the delta velocity and delta heading by playing the simulation forward. In the online component, when querying the learning algorithm for a specific $\hat{\mathbf{x}}_\mathbf{0}$, we use the desired delta velocity and delta heading instead.

The final feature vector $\hat{\mathbf{x}}$ could consist of any combination of these features. Automatic feature selection methods could be employed, but the feature vector used in the current implementation is simply chosen empirically. For the locomotion test scenario we chose only the body height features, as that set of features was found empirically to perform the best.

6.1.2 Pose dimensionality reduction

The high number of degrees of freedom in the simulated character together with the high number of spline segments in the control signal result in a very high-dimensional $\hat{\mathbf{y}}$. To make the learning problem faster to compute, we aim to find a better lower dimensional representation for $\hat{\mathbf{y}}$, which we call $\mathbf{y}$.

The vector $\hat{\mathbf{y}}$ is parameterized so that a separate number is used for each of the degrees of freedom of a given pose. When parameterized in this way, segments of $\hat{\mathbf{y}}$ can represent any pose within the joint range limits. However, this level of freedom is unnecessary or even undesirable for our purposes.

The majority of human motion can be explained as coordinated motion which controls multiple degrees of freedom at the same time. The simple parametrization of $\hat{\mathbf{y}}$ is thus highly redundant. We use unsupervised dimensionality reduction and the data set $\hat{\mathbf{Z}}$ to find a better parameterization $\mathbf{y}$. Additionally, if $\hat{\mathbf{x}}$ includes the joint angles, a lower dimensional representation

$\mathbf{x}$ can also be found.

Our choice of method is principal component analysis (PCA) [56] [33], for a number of reasons. First, PCA is a simple method to implement. Second, it is fast enough for online optimization. Third, PCA is applied using a linear transformation, the usefulness of which will become apparent in some of the details of the machine learning methods later in this chapter. Finally, and importantly, other authors have applied PCA successfully in the context of motion synthesis to reduce poses to as low as 5-10 dimensions [62].

To compute the principal components which form the basis in the reduced space, we gather the pose data available in both $\hat{\mathbf{X}}$ and $\hat{\mathbf{Y}}$ to a data matrix $\mathbf{P}$. The principal components are then computed using the usual method of finding the eigenvectors of the matrix $\mathbf{P}^T\mathbf{P}$. The number of principal components kept to form the new reduced basis is chosen empirically, though the number could also be chosen automatically e.g. by inspecting the data variance explained by each principal component.

After computing the principal components, transformation matrices $\mathbf{T}_{Y \leftarrow \hat{Y}}$ and $\mathbf{T}_{\hat{Y} \leftarrow Y}$ are formed. Thus, after the basis and the matrices have been computed offline, the data may be easily transformed online to and from the reduced space using simple matrix multiplication operations.

PCA successfully reduces the dimensionality from the original 34 degrees of freedom even down to the 5-10 dimensions mentioned earlier. Therefore, the machine learning methods described in the following sections all work in the reduced space Y, not the full space $\hat{Y}$. The sampling space used in online optimization is still the full space $\hat{Y}$, which means that the samples $\mathbf{y}$ received from the learning component must be transformed to $\hat{\mathbf{y}}$ before they can be used by the online optimizer.

6.2 Approximate nearest neighbors

The first machine learning method presented in this chapter is the simplest one. As it was already applied by Hämäläinen et al. [32] for sequential Monte Carlo motion synthesis, the approximate nearest neighbors (ANN) method serves as a baseline for the comparison to other methods.

We are given a data set of $(\mathbf{x}, \mathbf{y})$ pairs and we wish to estimate $\mathbf{y}$ for a given $\mathbf{x}_0$. The nearest neighbor method uses a simple but powerful idea: $\mathbf{y}$ of the pair $(\mathbf{x}, \mathbf{y})$ nearest to $\mathbf{x}_0$ is used as the estimate. The distance measure is the Euclidean distance between state vectors $||\mathbf{x} - \mathbf{x}_0||_2$, or equivalently, the sum of squared differences $||\mathbf{x} - \mathbf{x}_0||_2^2$.

The optimizer component looks for a single best $\mathbf{y}$ to apply to the simulated character. However, since we are using the machine learning component

to direct the optimization process, we can retrieve multiple $\mathbf{y}$ neighbors and use all of them for direction.

The K nearest neighbors found by the machine learning component, i.e. the control signals $\mathbf{y}_k$, are evaluated and weighted by the optimizer component. Just as other samples handled by the optimizer, the K nearest neighbors are used to form an implicit prior, which weighs the distribution of the generated samples. The found nearest neighbors are thus useful for the optimization component even if the optimal $\mathbf{y}$ may not be predicted.

A simple and slow way to find the K nearest neighbors is through a brute-force linear search on the whole data set. Unfortunately, for high-dimensional data sets, linear search is also the fastest known algorithm [55]. Luckily, fast approximate algorithms exist. Since we are interested only in creating an approximate prior for the optimizer, we can retrieve K approximate neighbors.

We integrate the open source library called Fast Library for Approximate Nearest Neighbors (FLANN), which is an implementation of the algorithm described by Muja and Lowe [55]. The details of the implementation are not highly relevant in this case, and they are outside the scope of this thesis. The amount of nearest neighbors searched K is an adjustable parameter which is exposed to the user of the system.

6.3 Locally weighted regression

Next, we will introduce a second machine learning method, called locally weighted regression (LWR) [13]. For efficiency, we will implement LWR on top of the approximate nearest neighbors method. As before, we first find K samples that are approximately nearest to the query variable $\mathbf{x}_0$. Unlike previously, we do not immediately evaluate the samples. Instead, we use the samples to form a local model, which will then be used for sample generation.

6.3.1 Model

The model is formed in the joint space

$$\mathbf{z} = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \tag{6.1}$$

i.e. we use both the control signal vectors and the state vectors to build the model.

The model in question is the multivariate Gaussian distribution:

$$\mathcal{N}(\mathbf{z}) = \frac{1}{\sqrt{(2\pi)^k |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{z}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{z}-\boldsymbol{\mu})\right). \tag{6.2}$$

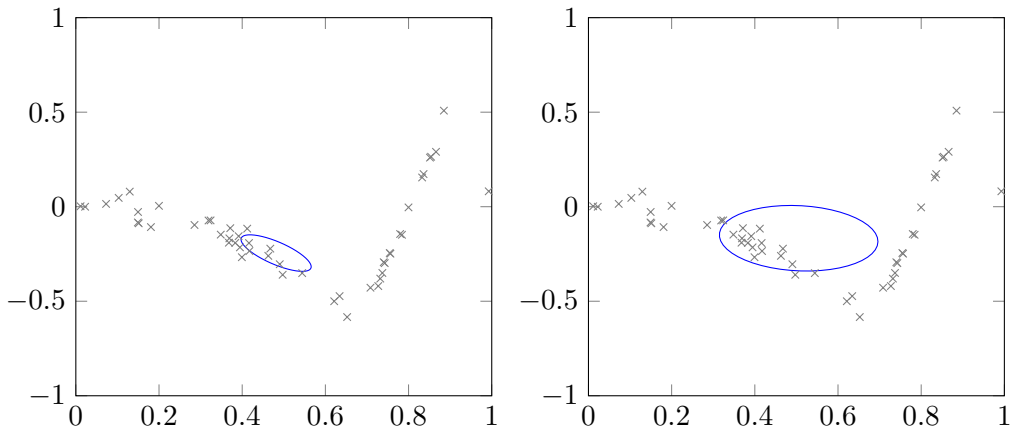


Figure 6.2: LWR on a synthetic data set at $x_0 = 0.5$. Two fitted models are shown for the same data set, one with $\sigma = 0.2$ (on the left) and one with $\sigma = 0.5$. In this case, the smaller deviation results in a better fit. The contours of the standard deviation for the Gaussian distributions are drawn in blue.

We describe the LWR model using a probabilistic approach, though the model may also be equivalently explained as an application of linear regression.

Figure 6.2 shows an example of LWR on a synthetic data set. Note that the actual $\mathbf{x}$ and $\mathbf{y}$ are high-dimensional vectors instead of the scalars depicted in the figure.

Some of the samples received from the ANN search are more useful than others. A kernel function weighs the relevance of the samples based on the distance to the query variable $\mathbf{x}_0$. We use a Gaussian kernel function

$$w(\mathbf{x}) = \exp(-\frac{||\mathbf{x} - \mathbf{x_0}||^2}{\sigma^2}) \tag{6.3}$$

in which the parameter σ^2 controls the width of the distribution. The value of the parameter can be chosen in many ways, for example by using cross-validation or by estimating a suitable value from the data. Since we can easily preview the performance of different values, we simply pick a constant value by hand.

The model parameters for the joint distribution are calculated using weighted least-squares. The parameters are partitioned as follows:

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_x \\ \boldsymbol{\mu}_y \end{bmatrix} \qquad \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{xx} & \boldsymbol{\Sigma}_{xy} \\ \boldsymbol{\Sigma}_{yx} & \boldsymbol{\Sigma}_{yy} \end{bmatrix}. \tag{6.4}$$

The model of the joint distribution is now fully described. For sampling,

we are given $\mathbf{x}_0$ and wish to compute $\mathbf{y}$. For this, we need the conditional distribution, which for the joint Gaussian distribution is also a Gaussian:

$$p(\mathbf{y}|\mathbf{x}_0) = \mathcal{N}(\boldsymbol{\mu}_{y|x}, \boldsymbol{\Sigma}_{y|x}). \tag{6.5}$$

The model parameters for the conditional distribution are simple to calculate, given the parameters of the joint distribution: [10]

$$\boldsymbol{\mu}_{y|x} = \boldsymbol{\mu}_y + \boldsymbol{\Sigma}_{yx}\boldsymbol{\Sigma}_{xx}^{-1}(\mathbf{x}_0 - \boldsymbol{\mu}_x) \tag{6.6}$$

$$\boldsymbol{\Sigma}_{y|x} = \boldsymbol{\Sigma}_{yy} - \boldsymbol{\Sigma}_{yx}\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\Sigma}_{xy}. \tag{6.7}$$

Optionally, ridge regression may be used by replacing $\boldsymbol{\Sigma}_{xx}$ with $\boldsymbol{\Sigma}_{xx} + \lambda\mathbf{I}$, where λ is the regularization parameter.

This is where the probabilistic formulation becomes advantageous. Instead of only finding the most likely control signal $\boldsymbol{\mu}_{y|x}$, we have estimated the probability of each control signal with the distribution $\mathcal{N}(\boldsymbol{\mu}_{y|x}, \boldsymbol{\Sigma}_{y|x})$. Any number of samples may now be generated from the conditional distribution.

The covariance matrix $\boldsymbol{\Sigma}_{y|x}$ guides the optimization more efficiently than the implicit prior formed by ANN. Unlike ANN, LWR uses the state vectors $\mathbf{x}$ of each sample to form the model. This helps to capture the relationship between $\mathbf{x}$ and $\mathbf{y}$, which leads to better estimates when $\mathbf{x}_0$ is distant from all of the training data.

Since the samples received from the ANN query are only used to form the model and are not evaluated for fitness, we may search for a larger amount of neighbors K. The relevance of the neighbors is weighed by the kernel function in Equation 6.3, rather than by the density of the data set, resulting in better generalization.

6.3.2 Sampling

We can sample from the conditional multivariate Gaussian distribution after we have computed the parameters in Equations 6.6 and 6.7. First, we compute the Cholesky decomposition

$$\boldsymbol{\Sigma}_{y|x} = \mathbf{L}\mathbf{L}^T \tag{6.8}$$

and generate a Gaussian distributed vector $\mathbf{z}$ with zero mean and unit variance. The full sample is then: [10]

$$\mathbf{y} = \boldsymbol{\mu}_{y|x} + \mathbf{L}\mathbf{z}. \tag{6.9}$$

Sampling in this way yields a sample from the correct Gaussian distribution, but we can reach an improvement if we examine the machine learning component together with the optimization component. Machine learning provides a probability distribution of $\mathbf{y}$ values based on the data computed offline, but the optimization component also estimates a probability distribution of $\mathbf{y}$. We can combine the information of the two distributions into a new distribution by computing their product.

The machine learning distribution given by LWR is a unimodal Gaussian, but the optimizer's distribution is a multimodal non-parametric distribution defined by the kd-tree, which is harder to handle mathematically. Instead of computing the product directly using the kd-tree, we compute the product with the proposal Gaussian (see line 23 of the algorithm on page 29) selected by the optimizer.

As before, the proposal Gaussian defines a region of the sampling space to explore based on the samples evaluated so far, but now, the LWR Gaussian helps to focus the search based on the training data. The generated samples still approximate the fitness function, but convergence is faster if the LWR Gaussian aligns with the dominant function mode.

In general, the product of two Gaussian densities is

$$\mathcal{N}(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) = c\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)\mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2), \tag{6.10}$$

but we can ignore the normalization factor c since we will only use the product for sampling. The parameters of the product distribution are: [58]

$$\boldsymbol{\mu}_c = (\boldsymbol{\Sigma}_1^{-1} + \boldsymbol{\Sigma}_2^{-1})^{-1}(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2) \tag{6.11}$$

$$\boldsymbol{\Sigma}_c = (\boldsymbol{\Sigma}_1^{-1} + \boldsymbol{\Sigma}_2^{-1})^{-1}. \tag{6.12}$$

To compute the parameters, we need to invert three matrices. One of these matrices, the covariance matrix of the proposal distribution, is diagonal and thus quick to invert. The remaining two have non-diagonal elements, though they could be restricted to be diagonal at the cost of worse estimates. The full covariance matrices have the dimensionality of the full sampling space Y, which means computing the inverses would be so slow that the advantages of product sampling would become irrelevant.

Fortunately, we can speed up the computation of the product by moving from the full sampling space Y to a reduced space R, and computing the product in R. We can can choose the reduced space R in such a way that the result of the product can be computed exactly. We will briefly explain why: the parameters $\boldsymbol{\mu}_{y|x}$ and $\boldsymbol{\Sigma}_{y|x}$ are formed by linearly combining the K samples received from ANN. Thus, the space spanned by $\boldsymbol{\mu}_{y|x}$ and $\boldsymbol{\Sigma}_{y|x}$

must have dimensionality of at most $K-1$. This spanned space is a good candidate for forming R, since the value $K-1$ is usually much lower than the dimensionality of the full sampling space.

Since we will need to transform both mean vectors and covariance matrices to R and back, we will find an affine transformation for the task. The result is stored in the augmented matrix:

$$\mathbf{T}_{R\leftarrow Y} = \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ 0,\ldots,0 & 1 \end{bmatrix}. \tag{6.13}$$

The vector $\mathbf{b}$ is easy to compute: it is simply $-\boldsymbol{\mu}_{y|x}$.

$\mathbf{A}$ is decomposed into:

$$\mathbf{A} = \left[\mathbf{a}_1,\ldots,\mathbf{a}_{N-1}\right]. \tag{6.14}$$

There are many ways to find the suitable basis vectors $\mathbf{a}_i$ that span the space. We choose to apply power iteration [43] successively to find the $K-1$ eigenvectors of the covariance matrix, i.e. the principal components. Using the principal components as the basis has the additional benefit that an approximation of the space R can be computed. We terminate the power iteration after 99% of the variance has been explained by the eigenvectors, or when 15 eigenvectors have been found. This approximation of R was empirically found to work as well as the exact R.

Using the matrix $T_{R\leftarrow Y}$, we can efficiently transform mean vectors and covariance matrices:

$$\boldsymbol{\mu}_R = \mathbf{T}_{R\leftarrow Y}\boldsymbol{\mu}_Y \tag{6.15}$$

$$\boldsymbol{\Sigma}_R = \mathbf{T}_{R\leftarrow Y}\boldsymbol{\Sigma}_Y\mathbf{T}^{-1}_{R\leftarrow Y}. \tag{6.16}$$

Furthermore, since some vectors and matrices are already being transformed from the uncompressed space $\hat{Y}$ to the PCA-compressed space Y, we can combine the two transforms into one matrix:

$$\mathbf{T}_{R\leftarrow \hat{Y}} = \mathbf{T}_{R\leftarrow Y}\mathbf{T}_{Y\leftarrow \hat{Y}}. \tag{6.17}$$

Despite these improvements to efficiency, generating samples using LWR is slower than when using other methods, though LWR is still useful since most of the time is still spent in evaluating the samples. The general reason for the efficiency issues is that the model is reconstructed for every new query variable $\mathbf{x}$.

6.4 Mixture of regressors

In this section, we will move from the non-parametric methods (ANN and LWR) to discuss parametric methods. Instead of storing every data point of the training set, our goal is to describe the entire training set with a model and a set of parameters. As a result, sampling should be faster than when using LWR, since the cost of constructing the model is moved to an offline training step. Furthermore, inspecting the whole training set instead of local approximations allows for potentially better estimates of the probability density.

6.4.1 Model

Mixture of regressors, or MoR, is a parametric regression model and a special case of the broader class of mixture of experts [35] models. The particular variant presented here is based on an application to computer vision by Agarwal and Triggs [2]. MoR fits a mixture of K Gaussians to the joint $(\mathbf{x}, \mathbf{y})$ space, modeling the data as

$$\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix} \simeq \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{6.18}$$

in which π_k are gating probabilities of each component k. The parameter K is either hand-picked or chosen using cross-validation.

The component distributions $\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ have a simplified structure. Within each component k, we assume a linear relationship between $\mathbf{x}$ and $\mathbf{y}$ with a Gaussian noise term $\boldsymbol{\epsilon}_k$:

$$\mathbf{y} = \mathbf{A}_k \mathbf{x} + \mathbf{b}_k + \boldsymbol{\epsilon}_k. \tag{6.19}$$

In other words, each component distribution is a single linear regressor for a portion of the $(\mathbf{x}, \mathbf{y})$ space. The full model is a weighted sum of these regressors, hence the name of the model.

Due to the imposed linear relationship, the mean for component k is:

$$\boldsymbol{\mu}_k = \begin{bmatrix}\boldsymbol{\mu}_{k,x}\\ \mathbf{A}_k \boldsymbol{\mu}_{k,x} + \mathbf{b}_k\end{bmatrix}. \tag{6.20}$$

The joint covariance matrix consists of the regression matrix $\mathbf{A}_k$, the marginal covariance matrix $\boldsymbol{\Sigma}_{k,xx}$, and the conditional covariance matrix $\boldsymbol{\Sigma}_{k,y|x}$:

$$\boldsymbol{\Sigma}_k = \begin{bmatrix}\boldsymbol{\Sigma}_{k,xx} & \boldsymbol{\Sigma}_{k,xx}\mathbf{A}_k^T\\ \mathbf{A}_k\boldsymbol{\Sigma}_{k,xx} & \mathbf{A}_k\boldsymbol{\Sigma}_{k,xx}\mathbf{A}_k^T + \boldsymbol{\Sigma}_{k,y|x}\end{bmatrix}. \tag{6.21}$$

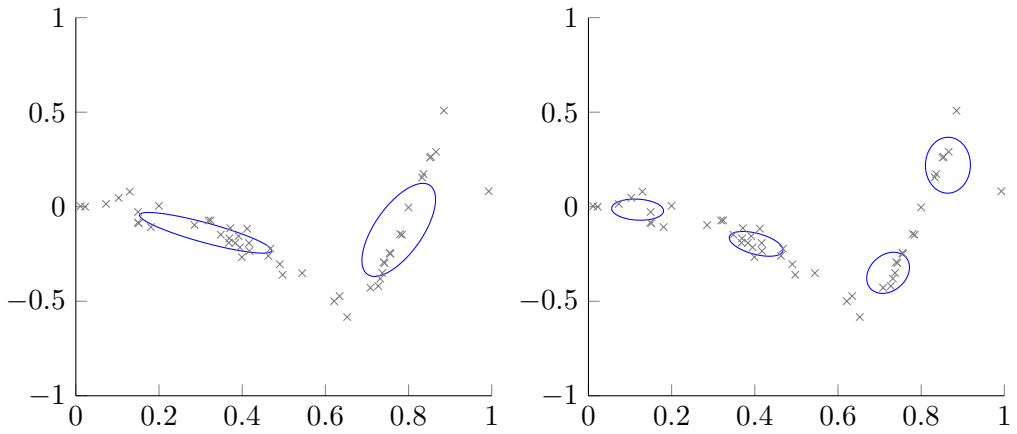


Figure 6.3: MoR on a synthetic data set. Two models are shown, $K = 2$ on the left and $K = 4$ on the right. The contours of the standard deviation for the component Gaussian distributions are drawn in blue.

Matrices $\mathbf{\Sigma}_{k,xx}$ and $\mathbf{\Sigma}_{k,y|x}$ are restricted to be diagonal to reduce the number of parameters. When m_x and m_y denote the dimensions of $\mathbf{x}$ and $\mathbf{y}$, each component distribution has $2m_x + 2m_y + m_x m_y$ parameters, whereas a Gaussian with a full covariance matrix has $m_x + m_y + \frac{1}{2}(m_x + m_y)(m_x + m_y + 1)$ parameters.

For a given state vector $\mathbf{x}_0$, we compute the relevance of each of the K mixtures and use the relevant regressors to estimate control signal $\mathbf{y}$. Additionally, we could estimate the probability of state vectors $\mathbf{x}_0$ in the training set using the marginal distributions for $\mathbf{x}$. This is currently not done in the system, but it could be used to estimate the usefulness of the training data for a given situation.

6.4.2 Training

Since there is no closed form solution for fitting the model to the training set, the training uses an iterative expectation-maximization (EM) [17] algorithm. The algorithm attempts to find the parameters $\boldsymbol{\mu}, \mathbf{\Sigma}$, and $\boldsymbol{\pi}$ which maximize the likelihood $p(\mathbf{Z}|\boldsymbol{\mu}, \mathbf{\Sigma}, \boldsymbol{\pi})$ of the whole joint training set $\mathbf{Z} = (\mathbf{X}, \mathbf{Y})$. We provide only a broad overview of the algorithm and do not attempt to prove its correctness.

In the EM algorithm, each data point in the training set has a corresponding component responsibility vector $\mathbf{r}_n$. The elements r_{nk} of the vector estimate the probability that component k was the mixture component that

generated the data point n.

In brief, the EM algorithm is as follows:

1. Initialize the component responsibilities $\mathbf{r}_n$ for all n.
2. M-step: Compute the component statistics $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$, and $\boldsymbol{\pi}$ using $\mathbf{r}_n$ as weights.
3. E-step: Compute all $\mathbf{r}_n$ using the new component statistics.
4. If converged, terminate, if not, go back to 2.

The algorithm is said to converge when the likelihood $p(\mathbf{Z}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi})$ no longer increases. Additionally, we terminate the algorithm if N iterations have passed.

The component responsibilities are initialized by running the k-means [49] clustering algorithm, using the k-means++ [6] variant. The initial clustering is performed in the joint space $(\mathbf{x}, \mathbf{y})$. We found empirically that initialization in only the $\mathbf{y}$ space worked nearly as well, but initialization in $\mathbf{x}$ alone resulted in a much poorer fit.

In the M-step, $\mathbf{A}_k$ and $\mathbf{b}_k$ are computed using weighted least squares regression. The result is stored in an affine matrix $\mathbf{J}_k$. For matrices weighted with matrix $\mathbf{W}$ constructed from the relevant component responsibilities $\mathbf{r}_{nk}$:

$$\begin{aligned} \mathbf{X}' &= \mathbf{W}\mathbf{X} \\ \mathbf{Y}' &= \mathbf{W}\mathbf{Y} \end{aligned} \tag{6.22}$$

we have the equation for $\mathbf{J}_k$:

$$(\mathbf{X}^T\mathbf{X}' + \lambda_1\mathbf{I})\mathbf{J}_k = \mathbf{X}^T\mathbf{Y}'. \tag{6.23}$$

The term $\lambda_1\mathbf{I}$ is added for regularization. The covariances of the residual matrix $\mathbf{Y} - \mathbf{X}\mathbf{J}_k$ form the conditional covariance matrix $\boldsymbol{\Sigma}_{k,y|x}$. The covariance matrix $\boldsymbol{\Sigma}_{k,xx}$ is simply estimated from the weighted marginal distribution of $\mathbf{x}$. For regularization, $\lambda_2\mathbf{I}$ may be added to $\boldsymbol{\Sigma}_{k,y|x}$ and $\boldsymbol{\Sigma}_{k,xx}$. The joint mean and covariance are computed using Equations 6.20 and 6.21.

Computing $\mathbf{r}_n$ in the E-step is mathematically straightforward:

$$r_{nk} = p(k|\mathbf{Z}_n, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) \tag{6.24}$$

$$p(k|\mathbf{z}, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \frac{\pi_k \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}. \tag{6.25}$$

However, in practice, $\mathcal{N}(\mathbf{Z}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ is very small, thus $\log(\mathcal{N}(\mathbf{Z}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k))$ is computed instead. The E-step is computationally intensive since the

Cholesky decomposition of $\mathbf{\Sigma}_k$ is required when evaluating the Gaussian density function. Luckily, the E-step is trivial to parallelize, as the $\mathbf{r}_n$ can be computed independently for each n.

The EM algorithm is prone to several degenerate cases and implementation difficulties, but instead of discussing them further, we refer to relevant literature [60] [10].

6.4.3 Sampling

To get control signal samples $\mathbf{y}$ for a state $\mathbf{x}_0$, we need the conditional distribution $\mathbf{y}|\mathbf{x}_0$. We can compute the conditional probability of each component almost as in Equation 6.25 of the training E-step, but this time only for marginal $\mathbf{x}$:

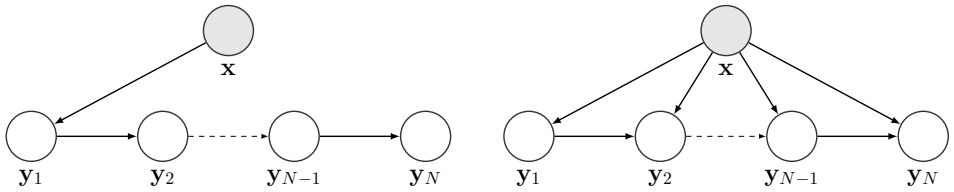
$$p(k|\mathbf{x}_0, \boldsymbol{\mu}, \mathbf{\Sigma}, \boldsymbol{\pi}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_0|\boldsymbol{\mu}_{k,x}, \mathbf{\Sigma}_{k,xx})}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_0|\boldsymbol{\mu}_{j,x}, \mathbf{\Sigma}_{j,xx})}. \tag{6.26}$$

To sample from the mixture of Gaussian distributions, we first sample a component k weighted by the discrete distribution given by $p(k|\mathbf{x}_0, \boldsymbol{\mu}, \mathbf{\Sigma}, \boldsymbol{\pi})$, then simply sample the conditional component distribution $\mathcal{N}(\mathbf{y}|\boldsymbol{\mu}_{k,y|x}, \mathbf{\Sigma}_{k,y|x})$. Sampling the component distribution can also be described as computing the regression result $\mathbf{y} = \mathbf{A}_k\mathbf{x}_0 + \mathbf{b}_k$ and adding Gaussian noise with covariance $\mathbf{\Sigma}_{k,y|x}$. Sampling from the component distribution is efficient since $\mathbf{\Sigma}_{k,y|x}$ is diagonal and independent of $\mathbf{x}$.

As previously with LWR, we want to sample the product of the optimization and machine learning distributions. For MoR, the product distribution is a little bit more complicated since both distributions are multimodal. As when sampling using LWR, we select a single proposal Gaussian, which simplifies the problem to a product of the proposal Gaussian and the mixture of Gaussians given by MoR.

In general, the product of a single Gaussian with a Gaussian mixture of K components is another Gaussian mixture with K components. The product can be approximated or computed exactly. Efficient approximations have been developed, as the the problem has some applications, for example in nonparametric belief propagation. [34]

Our approximation is simple: we pick a random component from the mixture and calculate its product with the single Gaussian. The approximation works well in practice, though it does cause additional bias. We suspect that the approximation works well because the distributions often contain only a few modes. We also implemented exact sampling which calculates the products for each mixture component, but the implementation was discarded as it failed to improve the sampling and added a small performance cost.



(a) Equation 6.32 (b) Equation 6.33

Figure 6.4: Bayesian networks corresponding to the factorized distributions.

6.4.4 Factorized mixture of regressors

So far, we have treated the control signals $\mathbf{y}$ as plain data with no regard to its structure. If we can find some exploitable coherence in the data, we can improve the model by reducing the number of learning parameters.

The control signal consists of N spline segments:

$$\mathbf{y} = \left[\mathbf{y}_1^T, \ldots, \mathbf{y}_N^T\right]^T. \tag{6.27}$$

We know that the segments $\mathbf{y}_i$ are closely related to each other. For example, if we have two control signals $\mathbf{y}$ and $\boldsymbol{\gamma}$, and for any two segments i and j we have

$$\mathbf{y}_i \approx \boldsymbol{\gamma}_j, \tag{6.28}$$

then it is likely that also

$$\mathbf{y}_{i+1} \approx \boldsymbol{\gamma}_{j+1} \tag{6.29}$$

for the next segments.

To use this dependence to our advantage, we make two assumptions. First, for $i > 1$:

$$p(\mathbf{y}_i|\mathbf{y}_{i-1}, \mathbf{x}) = p(\mathbf{y}_i|\mathbf{y}_{i-1}). \tag{6.30}$$

In other words, $\mathbf{y}_i$ is conditionally independent of $\mathbf{x}$, given $\mathbf{y}_{i-1}$. The second assumption, for $i > 1$ and $j > 1$:

$$p(\mathbf{y}_i|\mathbf{y}_{i-1}) = p(\mathbf{y}_j|\mathbf{y}_{j-1}). \tag{6.31}$$

Using these assumptions, we can factorize the probability into

$$p(\mathbf{y}|\mathbf{x}) = p(\mathbf{y}_1|\mathbf{x})p(\mathbf{y}_2|\mathbf{y}_1)p(\mathbf{y}_3|\mathbf{y}_2)\ldots p(\mathbf{y}_N|\mathbf{y}_{N-1}) \tag{6.32}$$

where only the conditional distributions $p(\mathbf{y}_1|\mathbf{x})$ and $p(\mathbf{y}_N|\mathbf{y}_{N-1})$ are distinct, that is, we do not need a separate distribution for every factor of the product.

Both conditional distributions are estimated with the mixture of regressors model as in the unfactorized case. Even if the assumptions in 6.30 and

6.31 do not hold for all $\mathbf{y}$, we get a reasonable estimate since the mixture of regressors model can represent multimodal densities. For simplicity, the two distributions share the same amount of components K.

The amount of learning parameters is reduced drastically, resulting in less variance for the same amount of training data. The obvious drawback is the potentially increased bias caused by the strong assumptions.

The probability could have been factorized in other ways, too. For example

$$p(\mathbf{y}|\mathbf{x}) = p(\mathbf{y}_1|\mathbf{x})p(\mathbf{y}_2|\mathbf{y}_1,\mathbf{x})p(\mathbf{y}_3|\mathbf{y}_2,\mathbf{x})\dots p(\mathbf{y}_N|\mathbf{y}_{N-1},\mathbf{x}) \tag{6.33}$$

would allow for some dependence in the input $\mathbf{x}$, which would remove some bias at the cost of extra model parameters.

The factorized distribution in Equation 6.32 can be sampled using ancestral sampling [8]: first sample the distribution $p(\mathbf{y}_1|\mathbf{x})$ for the known $\mathbf{x}$, then $p(\mathbf{y}_2|\mathbf{y}_1)$ for the sampled $\mathbf{y}_1$, etc.

To sample from the product distribution, we may again retrieve a single proposal Gaussian from the optimizer. We apply a similar factorization to the distribution q represented by the proposal Gaussian:

$$q(\mathbf{y}|\mathbf{x}) = q(\mathbf{y}_1|\mathbf{x})q(\mathbf{y}_2|\mathbf{y}_1,\mathbf{x})q(\mathbf{y}_3|\mathbf{y}_2,\mathbf{x})\dots q(\mathbf{y}_N|\mathbf{y}_{N-1},\mathbf{x}) \tag{6.34}$$

which is

$$q(\mathbf{y}|\mathbf{x}) = q(\mathbf{y}_1)q(\mathbf{y}_2)q(\mathbf{y}_3)\dots q(\mathbf{y}_N) \tag{6.35}$$

since the distribution is diagonal. The full product of Equations 6.32 and 6.35 can be computed using ancestral sampling per segment, that is, first sampling $p(\mathbf{y}_1|\mathbf{x})q(\mathbf{y}_1)$, then using the sample as $\mathbf{y}_1$ to compute $p(\mathbf{y}_2|\mathbf{y}_1)q(\mathbf{y}_2)$, etc. Each segment computed in this scheme is the product of a Gaussian and a mixture of Gaussians, which can be computed just as previously in the unfactorized case.

Though the ancestral sampling method above produces unbiased samples, the sampling method used in our system works slightly differently when computing the conditional $p(\mathbf{y}_2|\mathbf{y}_1)$. Instead of using the sampled $\mathbf{y}_1$, we use the mean of the Gaussian component in the mixture $p(\mathbf{y}_1|\mathbf{x})$ which generated the sampled $\mathbf{y}_1$. Although we have only improved results as justification for the scheme, we suspect using the mean instead of the sample helps in regularizing the generated samples.

6.5 Self-organizing map

Previously, we have directed the optimization process towards likely control signals by placing a probabilistic prior on the sampling space Y. In this

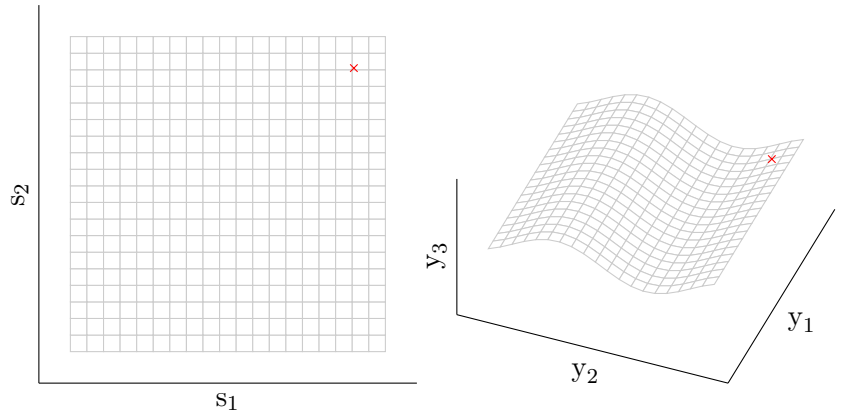


Figure 6.5: A geometric interpretation of the self-organizing map. The SOM grid is shown in two spaces: on the left, the reduced space S; on the right, the full space Y. The red cross shows how a point in the reduced space is mapped nonlinearly to the full space.

section, instead of optimizing the control signals in Y, we optimize in a reduced space S, where each point $\mathbf{s} \in S$ has a corresponding viable control signal $\mathbf{y} \in Y$. To find the space S, we apply unsupervised learning on the $\mathbf{Y}$ data set.

Our method of choice is the self-organizing map [39], which was first introduced as a computer simulation of a neural network model. The SOM algorithm finds a mapping from the reduced space S to the full space Y, while preserving the topology of S in Y.

The previous two methods in this chapter, LWR and MoR, were defined using a probability density model. SOM, on the other hand, is more accurately defined as an algorithm rather than a model: there is no simple probabilistic interpretation of SOM. We could have also approached the problem of finding the reduced space S using probabilistic tools, for example using generative topographic mapping (GTM) [11], a probabilistic model based on SOM.

The probabilistic tool set used by GTM is very similar to the one used earlier in this chapter: it is based on a mixture of Gaussians and it is trained using an expectation-maximization algorithm. Despite the differences in form, GTM is claimed to perform very similarly to SOM. Both SOM and GTM define a nonlinear relationship between Y and S and both are sufficiently efficient to compute. While using GTM instead of SOM would have had some advantages, we chose to implement SOM because of its simplicity.

In SOM, the data points are assumed to lie on a low-dimensional manifold

S, which is embedded nonlinearly in the high-dimensional space Y. The manifold is represented as a grid in which each grid cell has a corresponding point in the high-dimensional space Y. The grid cells are sometimes called nodes or neurons. In Figure 6.5 we provide a simplified geometrical view of the type of mapping.

The topology of the grid in S can be defined in many ways. The grid commonly consist of either hexagonal or rectangular cells. At the borders of the grid, the cells may be connected to the cells on the other side of the grid, resulting in cylindrical and toroidal topologies. The space S could have any dimensionality, but is most often only two-dimensional, which allows for simple visualizations. Our implementation uses a two-dimensional rectangular grid.

The SOM algorithm finds the embedding by iteratively moving the manifold in place, i.e. by optimizing the weight matrix $\mathbf{W}$ in which the Y space positions of the points corresponding to the grid cells are stored. The specific SOM training algorithm implemented is based on Haykin's variant [29]. The full algorithm is as follows:

1. Construct the initial weight matrix $\mathbf{W}(0)$, set $n = 0$.
2. Choose a random training sample $\mathbf{y}(n)$ from the training set.
3. Find the column $\mathbf{W}_i(n)$ nearest to the training sample $\mathbf{y}$ using the Euclidean distance.
4. Update each column $\mathbf{W}_j(n+1)$ within the topological neighborhood $h_{j,i}(n)$, using the formula:

$$\mathbf{W}_j(n+1) = \mathbf{W}_j(n) + \eta(n)h_{j,i}(n)(\mathbf{y}(n) - \mathbf{W}_j(n)). \tag{6.36}$$

5. Set $n = n + 1$, go to 2 unless N iterations have passed.

For initialization, we first use the training set $\mathbf{Y}$ to find the two principal components $\mathbf{v}_1$ and $\mathbf{v}_2$ with the largest corresponding eigenvalues λ_1 and λ_2. We then set $\mathbf{W}$ columns by drawing random samples from the space spanned by $\mathbf{v}_1$ and $\mathbf{v}_2$ using λ_1 and λ_2 as variances. One common initialization method is to sample the vector components completely randomly. The algorithm will manage to find a good result even with the random initialization method, but for fast convergence, other methods should be chosen. [40]

The topological neighborhood is defined using the Gaussian function

$$h_{i,j}(n) = \exp(-\frac{||\mathbf{s_i} - \mathbf{s_j}||^2}{\sigma(n)^2}) \tag{6.37}$$

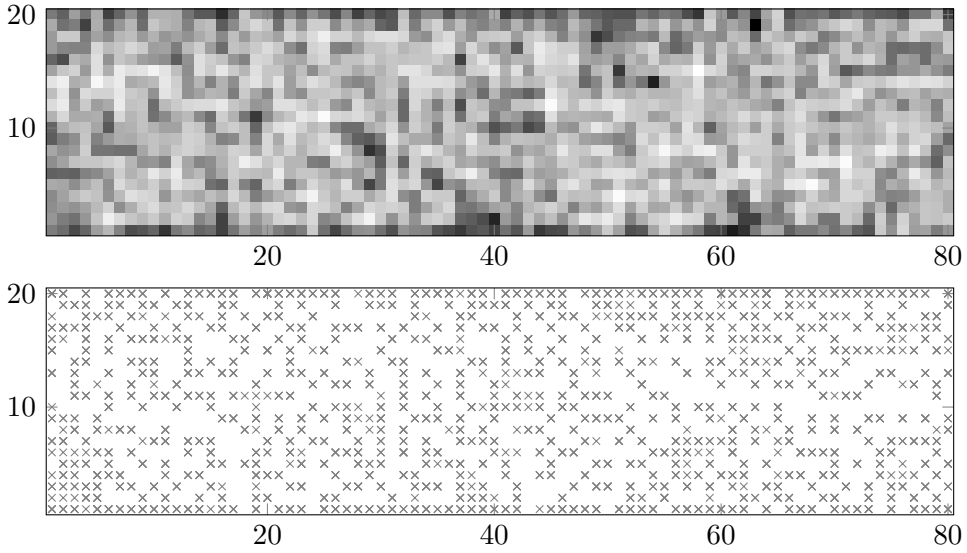


Figure 6.6: Self-organizing map of a run cycle data set. The horizontal axis wraps around at the edges, i.e. the map is cylindrical. On top, we show the U-matrix in which darker colors correspond to higher distances between grid cells. On the bottom, each training sample is drawn in the grid cell corresponding to the nearest point in Y.

which has a time-varying kernel width

$$\sigma(n) = \max(\theta_{min}, \theta_0 \exp(-\frac{n}{t_1 M})) \tag{6.38}$$

where θ_{min}, θ_0, and t_1 are adjustable parameters and M is the amount of training samples in the training set. For the learning rate we use:

$$\eta(n) = \max(\eta_{min}, \eta_0 \exp(-\frac{n}{t_2 M})), \tag{6.39}$$

with similarly η_{min}, η_0, and t_2 as adjustable parameters.

The results of the SOM algorithm are commonly visualized using the U-matrix. Each cell in the U-matrix depicts the average distance from $\mathbf{W}_i$ to neighboring grid cells $\mathbf{W}_j$. Figure 6.6 shows an example on real data.

SOM defines only a discrete points in S, but the sequential Monte Carlo optimization algorithm was defined for a continuous space. We make the discrete space continuous simply by using bilinear interpolation on the nearest four grid cells. The SOM mapping has high bias: it can only generate samples in Y that are located on the two-dimensional manifold.

In the SOM algorithm, any custom type of distance measure can be defined. The Euclidean distance used compares the components in the spline parameterization, which is not ideal since similar resulting signals may have different parameterizations. One potential type measures the frame-by-frame distance between the control signals created by the splines. We implemented both, but in practice there was no difference between the two.

We have described SOM for the space Y using samples of $\mathbf{y}$ and have so far ignored the $\mathbf{x}$ in the training data. It would be possible to treat SOM as only as an unsupervised preprocessing step and use the $\mathbf{y}|\mathbf{x}$ mapping on top for supervised learning. However, after a quick unsuccessful test using ANN for the $\mathbf{y}|\mathbf{x}$ mapping, we chose not to pursue this idea further.

6.6 Summary

In this chapter, we have presented several different approaches for solving the machine learning problem. We started with supervised learning methods by explaining the non-parametric approximate nearest neighbors (ANN) method. With locally weighted regression (LWR), we posed the problem probabilistically and introduced the concept of product sampling.

From local methods, we moved to the global mixture of regressors (MoR), which was used to parametrically estimate the probability density of the joint space. The factorized mixture of regressors (FMoR) method was used as an example of how the number of parameters in the model may be reduced using more detailed modeling of the joint probability density. Finally, the unsupervised self-organizing map (SOM) algorithm was used to reduce the sampling space to a lower dimensionality.

We chose to implement relatively simple methods that can be used in the nonlinear, multivariate, and multimodal problem without running into scalability problems. As mentioned earlier in this chapter, the machine learning literature names a multitude of different models and algorithms that have been used in various learning problems. However, in the scope of this thesis, we cannot evaluate all the possible choices. For example, other interesting methods include the relevance vector machine (RVM) [69], its multivariate extension MVRVM [68], and random forests [12].

There is room for improvement in other parts of the machine learning component as well. For example, different features could have been selected, automatic feature selection or feature extraction methods could have been experimented with, etc.

In the next chapter, we will present test results that help to evaluate how successful our choices were.

Chapter 7

Evaluation

In this chapter, we will test how well the implemented data-driven sequential Monte Carlo system meets our goals. In the first section, we will see how well the system is able to track reference animations, i.e. we evaluate the offline component of the system. In the second section, the online component is evaluated, and the implemented machine learning methods are compared.

7.1 Offline component

Reference animation tracking was tested on a diverse set of reference animations depicting various human motion tasks. Most of the animations were created using motion capture, though some keyframed clips were tested as well. The motion capture data was in general not properly cleaned: in some of the clips feet penetrate the ground and in some clips limbs jitter because of noise.

The data consisted of both low-energy tasks such as walking or standing in place and of high-energy tasks such as running or jumping. Most of the data depicted locomotion, but in some clips the character was off her feet: for example, some clips had the character getting up from a prone position. Some clips showed agile or difficult movements such as flips or jump kicks.

Table 7.1 shows the parameters used in the fitness function. The parameters were adjusted by hand, and the same set of values was used for all the reference animations tested.

By default the optimizer generates 250 samples per frame, which means that 2-3 frames may be optimized per second on a standard desktop computer. Though 250 samples was found adequate for tracking, the sample count was increased to 500 when generating data for the online component so that the full length of the generated control signal would be of high qual-

Table 7.1: Parameters used for tracking.

Parameter	Value	Unit	Difference measured
σ_{a1}	0.05 (2.9)	rad (deg)	Joint angles (average cost)
σ_{v1}	0.02 (1.1)	rad/s (deg/s)	Joint velocities (average cost)
σ_{head}	0.05 (0.29)	rad (deg)	Head orientation
σ_{a2}	0.2 (11)	rad (deg)	Joint angles (terminal cost)
σ_{v2}	0.1 (5.7)	rad/s (deg/s)	Joint velocities (terminal cost)
σ_{root}	0.1 (5.7)	rad (deg)	Root orientation
σ_{rvel}	0.1 (5.7)	rad/s (deg/s)	Root angular velocity
σ_{end}	0.1	m	End effector heights
σ_{bal}	0.1	m	Balance vectors
σ_{bvel}	0.5	m/s	Center of mass velocity

ity. The optimizer could not in general be used for real-time tracking, except for easy motions where the character had both of her feet locked in place at all times.

Figure 7.1 shows captured frames of successfully tracked motions. The system was able to follow the various motions in the test set, despite small errors and noise in the reference animations.

In general, the system tracked animations robustly, but the quality of the tracked motions was often inadequate. Sometimes parts of the animation, such as foot contacts, were not reproduced in the tracked animation, though the larger scale movements were captured correctly.

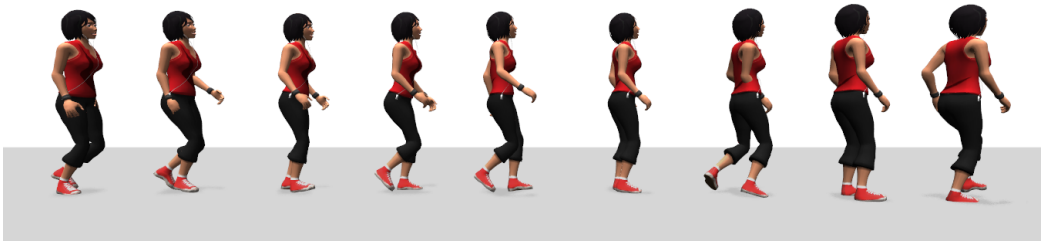
The quality of high-energy movements was perceptually better than that of low-energy movements. For example, a slow walk resulted in a jittery movement, but a lively jogging motion could be tracked at nearly the quality of the original motion.

Most motions synthesized could be characterized as discontinuous or jittery. When slowly walking forward, the character does not seem to control her limbs purposefully to generate forward momentum, but instead at a rapid rate adjusts her pose to match the reference motion.

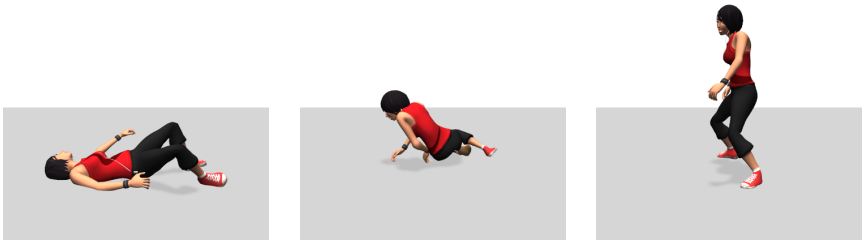
7.2 Online component

We used the locomotion application introduced in Chapter 4 to evaluate the online component. In the standard test setup, the character runs on top of a flat ground plane at a target speed towards a target direction. Only a single short (0.8 seconds) looping animation clip of the character running straight

(a) Jogging.



(b) Walking and occasionally stopping.



(c) Getting up from the floor.

(d) A flip.

Figure 7.1: Results of tracking reference animations.

Table 7.2: Parameters used for dimensionality reduction and machine learning methods in the locomotion test case.

Method	Parameter	Description
PCA	$d = 10$	Principal components kept
FLANN	$K = 5$	ANN samples retrieved per frame
LWR	$K = 30$	ANN samples retrieved per frame
	$\sigma = 0.0001$	Kernel width
	$\lambda = 0$	Regression regularization
MoR / FMoR	$K = 80$	Number of classes
	$\lambda_1 = 0.0001$	Regression regularization
	$\lambda_2 = 10^{-13}$	Covariance regularization for $\mathbf{x}$
	$N = 50$	EM iterations
SOM	$W = 80$	Grid width
	$H = 20$	Grid height
	$N = 1000$	Training algorithm iterations
	$\sigma_0 = 100$	Initial neighborhood width
	$t_1 = 50$	Neighborhood width decrease scale
	$\sigma_{min} = 10^{-15}$	Minimum neighborhood width
	$\eta_0 = 0.1$	Initial learning rate
	$t_2 = 100$	Learning rate decrease scale
	$\eta_{min} = 0.01$	Minimum learning rate

was used as reference data, but the same clip was tracked multiple times to generate more samples. The total training set consisted of 6900 samples.

Since our aim is to create a controller for an interactive application, the low amount of only 24 samples is generated per frame. With this sample count, interactive frame rates are achievable on a standard desktop computer regardless of the type of machine learning method used. We will discuss performance in detail in Subsection 7.2.2.

We tested for differences between the machine learning methods introduced in the previous chapter. Each method includes a number of adjustable parameters, which were tuned before running the experiments. Most parameters were simply hand-picked by adjusting the values and measuring the character's performance, but for the mixture of regressors models cross-validation was used to look for optimal parameters. The chosen parameters are summarized in Table 7.2.

7.2.1 Robustness

To evaluate robustness, we measure the time to failure: the duration the character can keep running without falling down or stopping. The time to failure could also be interpreted as the inverse of the probability of failure per second. To detect the moment of failure, we simply measure the fitness of the optimized sample and trigger failure when the fitness drops under a minimum threshold.

Alternatively, we could have chosen the average fitness over a fixed time horizon as the measure. However, we chose time to failure instead, as though the average fitness can be used to compare the quality of samples, it does not accurately measure how well the different methods generalize to new situations.

The experiment is initialized by setting the character in a running pose with appropriate initial velocities. After each failure, the experiment is restarted by returning the character back to the initial pose. A total of 50 of these experiments are run for each machine learning method and the average durations of the experiments are compared.

To make the test more challenging, the character is given a new random target direction every 5 seconds. Since no examples of turning are given, the controller must be able to generalize the forward running motion into synthesized turns.

Figure 7.2 shows the results of the machine learning method comparison. The self-organizing map (SOM) is the worst performer with under half a minute of running on average. Additionally, when SOM was used, the character had trouble running straight towards the target direction and ran along an S-shaped curve instead. Most methods kept the character running for 4-5 minutes, but factorized mixture of regressors (FMoR) managed about 8 minutes.

Additionally, we took the best performing machine learning method FMoR and evaluated the importance of the system's components by disabling them one at a time. The evaluated components were as follows:

Product sampling. Instead of sampling from the product of the optimization and machine learning distributions, we drew 5 samples directly from the machine learning distribution as with ANN. The remaining samples were generated using the kd-tree alone without the machine learning distribution.

Previous best sample. The best sample of the previous frame was not wound and used in the next frame.

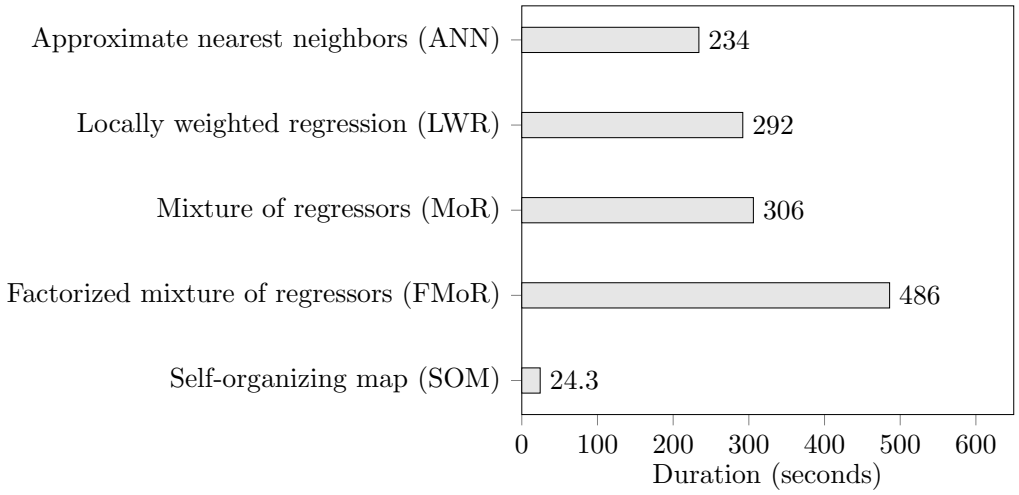


Figure 7.2: Performance of the compared machine learning methods in the robustness test. The length of the bar represents the average time the character ran in the experiment before falling down or stopping. Details can be found in the text.

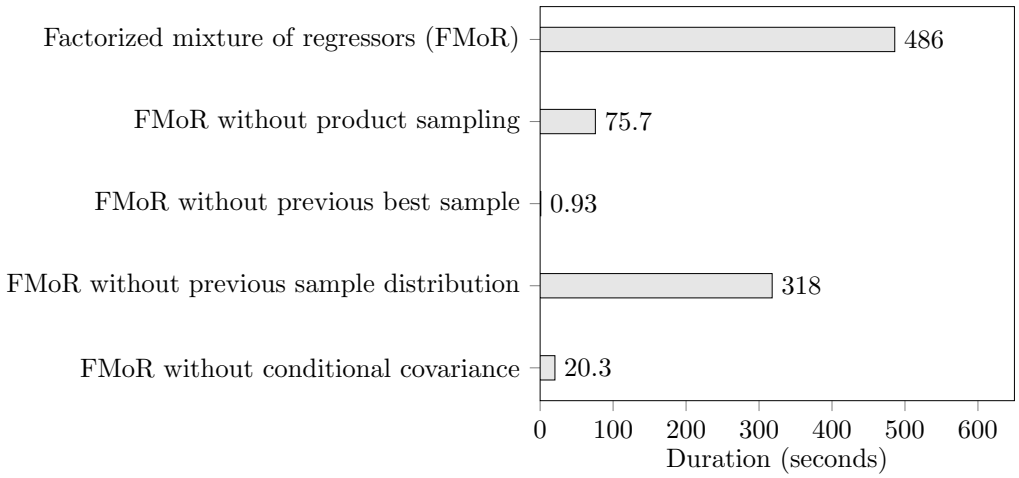


Figure 7.3: The effect of disabling different components of the system.

Previous sample distribution. The pruned distribution of the samples in the previous frame was not used as a prior for the next frame.

Conditional covariance. Instead of sampling from the conditional mixture of Gaussians distribution, we took the best conditional regressor component and used its mean without the covariance as the estimate.

The results can be seen in Figure 7.3. Perhaps surprisingly, leaving out the previous best sample results in an almost complete failure to synthesize a running motion. This suggests that the fitness function modes corresponding to the best samples are so narrow, that the pruned kd-tree from the previous time step is not alone sufficient to keep track of the modes.

We also tested robustness against external disturbances by pushing the player at chest level from different directions. With only the training data for a straight undisturbed run, the character was able to recover more than 90% of the time from 200 N pushes applied for 0.1 seconds.

The system's ability to adapt to new environments was tested with two types of terrain. The first test had a constant slope which the character ran up or down. The character was able to run on slopes of about 5 degrees, with generally better performance when running down the slope. The second test had the character run in different directions on a bumpy surface. The system worked quite robustly on the bumpy terrain, but the motion quality was decreased.

7.2.2 Performance and sample counts

The computer used to run the tests had an Intel Xeon E3-1230 V2 CPU with 4 physical cores and 8 logical cores. The evaluation of the samples is multithreaded, but the current implementation uses some unnecessary locking, which leads to only about 80% of the CPU to be utilized. With 24 samples per frame, the interactive locomotion test case reached a rate of about 20-24 frames per second depending on the machine learning method used.

We tested how increasing the amount of samples generated per frame affects the robustness by measuring the average fitness over a 60 second interval. Figure 7.4 shows the results: the effect of increased samples is significant up to about 50-100 samples per frame. The amount of samples generated for the interactive test case is clearly small, as the fitness plummets when fewer samples are generated.

In Figure 7.5, we compare the computational cost of the machine learning methods in relation to the physics simulation cost. SOM is not included from the comparison, as it has no initial cost, and drawing samples has a cost too small to accurately measure.

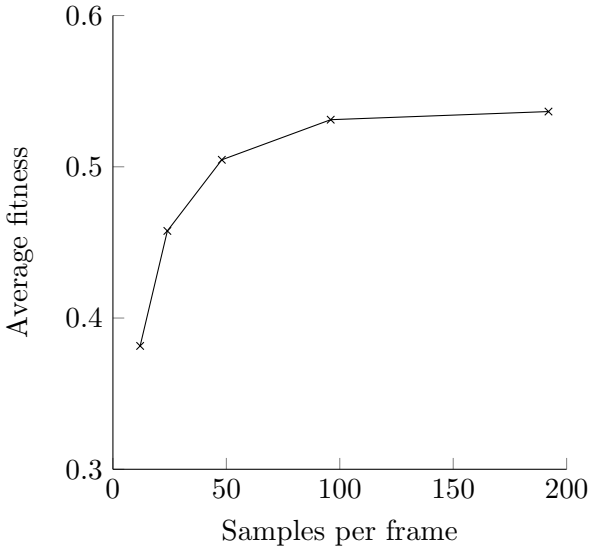
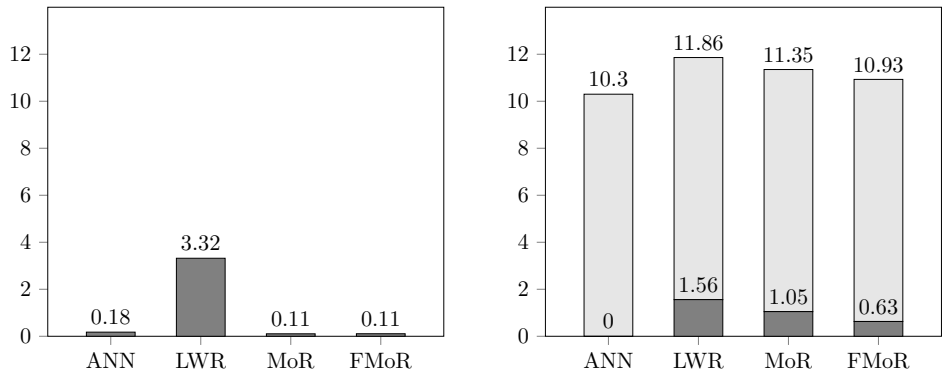


Figure 7.4: Average fitness of the best sample as a function of samples per frame when using factorized mixture of regressors. The points correspond to 12, 24, 48, 96 and 192 samples generated.



(a) Initial cost (milliseconds) (b) Per sample cost (milliseconds)

Figure 7.5: Computational cost of the machine learning methods in relation to physics simulation cost. On the left, the bars represent the time taken to compute $\mathbf{y}|\mathbf{x}_0$ for new $\mathbf{x}_0$. On the right, the lengths of the bars represent the time it took to draw (darker bar) and simulate (lighter bar) a single sample. See the text for further analysis.

Though LWR, MoR, and FMoR have higher per sample costs than ANN, their contribution to the total per sample cost is not significant, as most of the time is spent on evaluating the samples. The higher initial cost of LWR is highlighted in the locomotion test case, since the amount of samples drawn is relatively small. Additionally, since the computation resulting in the initial cost cannot be easily performed in parallel, in our simple test case the CPU usage is lowered.

Training the MoR and FMoR models is relatively quick, both requiring 2-3 minutes of computation for the locomotion training set. Training SOM is significantly slower, requiring 20-30 minutes, though we suspect that the amount of iterations could be lowered without severely affecting the quality of the trained model.

7.2.3 Motion quality

In general, motion quality in the online test case suffers from the same problems as the offline case: the motion is at worst discontinuous and jittery. However, since the simple test case presented here uses an easily trackable lively jogging reference animation, the synthesized motion is mostly of acceptable quality.

The quality problems increase when the controller has to generalize to motions not found in the training set. For example, running on uneven terrain greatly increases the jitter in the movement compared with running on flat ground.

The choice of machine learning method did not in general noticeably affect the quality of motion. The only exception was the previously mentioned problem with the self-organizing map, as its generally poor performance made the character unable to run in the target direction.

Some tests were run with an increase of simulation frequency to 50 Hz from the original 30 Hz. The frequency change did help with the robustness and especially with the motion quality, but since the optimization and simulation costs per frame were increased, the simple locomotion test case could no longer be run at real-time frame rates.

Additional tests were run in hopes to decrease the jitter in the movement by using a stricter smoothness target in the fitness function. Unfortunately the stricter smoothness goal greatly decreased the robustness and had only a small impact on the quality of the motion.

7.3 Analysis

The experiments were performed to both evaluate the developed data-driven sequential Monte Carlo motion synthesis framework and to compare the implemented machine learning methods. First, we will look analyze results for the former.

The results show that the sequential Monte Carlo optimizer can be used to robustly track various kinematic reference animations using a high-dimensional character. The motion quality of the synthesized motions is often poor, but we consider this a problem caused by the simple biomechanical model, rather than an innate feature of the optimizer used.

Despite the problems, our system bears comparison to state of the art tracking methods. For example, Liu et al. [48] have developed a similar sampling-based system for tracking. They are also able to track various types of motion, with roughly comparable robustness. Based on the published sample animations, their motion quality is higher, though they report similar problems with smoothness and stiffness, and have a slightly more complex biomechanical model. The biggest difference to their work is in computational cost: their sampling strategy allows tracking at 1/25 of real time using a 80-core computer cluster. In comparison, our system tracks animations at 1/10 to 1/15 of real-time on a single 4-core computer.

The online controller also uses sequential Monte Carlo successfully: it is able to synthesize motions for turning, slopes, and bumpy terrain in real-time, given only a single short reference motion. Many other authors have developed similar systems that use low-dimensional planning or various task-specific simplifications to synthesize walking or running motions. By contrast, our system performs planning using the full physical character with all the degrees of freedom and only a high-level fitness function to define the motion.

The results of the locomotion test case are perhaps closest to the work of Kwon and Hodgins [44], who also use trajectory optimization with reference motion data. Their system has similar generalization capabilities for turning, pushes, and slopes, and both systems can perform at interactive frame rates. However, their system uses low-dimensional planning, and depends on some low-level features, such as footsteps.

The locomotion controller by Muico et al. [53] also shows similar results and has some similarities in implementation. The quality of the synthesized motions generated by our system is notably worse, though our interactive controller is much more responsive, and our system is better at handling external disturbances and changes in the environment.

Our system is not able to match some of the more recent work in terms of robustness or agility. For example, Han et al. [27] exhibit robust responses to disturbances of up to 500 N, compared with our 200 N. However, direct comparisons are difficult to make, as details about the physical models and the reference motions are not available.

Although we present our implementation as a general-purpose framework for online motion synthesis, we only provide results for a simple locomotion test case using a single reference animation. Furthermore, the test case is a relatively easy one, requiring only very limited deviation from the original motion.

Some further tests were performed by adding turning motions into the training set. In general, the turning motions worked in the online locomotion test case, but unexpectedly, the controller had trouble using the turning motions together with the straight running motions, as the two motions seemed to form two disjoint sets in the training data. We suspect that the problem could be alleviated by precomputing transitions between the two sets, similarly to Muico et al [53].

In the machine learning comparison test, most of the more complex methods outperformed the ANN baseline. The only exception is SOM, which performed significantly worse. We suspect that the high bias induced by the restriction to a reduced two-dimensional sampling space is the cause for the failure, rather than the specific reduction method used. Excluding SOM, the differences between the machine learning methods were clearly measurable but relatively small, with the best method (FMoR) achieving twice the robustness of ANN.

In practice, greatly increasing the complexity of the system to double the robustness might not be justified. However, we view the results as confirmation for some generic ideas for future work, rather than a fully developed system. For example, the results show that modeling the training data as a probability distribution and performing sampling using the product with the optimizer's distribution seem to increase the efficiency. Additionally, factorizing the probability distribution per spline segment also increases efficiency, which suggests that the simpler model does not induce significant extra bias.

Chapter 8

Conclusions and future work

In the beginning of this thesis, we set out to extend the sequential Monte Carlo motion synthesis framework to use prior knowledge. We named two potential advantages to attain: efficiency and quality. Did we succeed?

8.1 Conclusions

We will start with the aspects of the implemented system that did work. For one, the reference animation tracking method proposed is able to robustly track various types of motion. The tracking is simple to use, as the parameters need not be adjusted per animation. The performance of the system is adequate, though not real-time.

We were able to use the results of the tracking to direct the online optimization process. The system was able to reproduce the tracked running animation in real-time and could generalize the data to synthesize motion for turning and running on uneven terrain. As the fitness function only describes a high-level velocity goal, the system is not specific to one style or type of motion.

We compared various machine learning methods more sophisticated than the previously implemented approximate nearest neighbors. Most of the new methods did improve the efficiency in the test case, though the improvement was relatively small. We gathered evidence to support some ideas for the learning component: probabilistic modeling of the training set, sampling from the product distribution, and factorizing the distribution by segments.

In summary, we claim that the efficiency goal was met, but the same cannot be said about the quality goal. The synthesized motions are at best passable and at worst jittery and inadequate. Furthermore, compared with kinematics-based approaches, the animator using the system has less control:

although different kinds of reference animations can be used to produce different kinds of motion, the animator has only indirect control over the results. We suspect that the quality goal is not attainable without improvements to the biomechanical modeling of the character.

Some aspects of the system are still left unknown. For example, the locomotion test case used to evaluate the system was very limited, so it is difficult to evaluate how well the system would work with different types of animation and with different motion tasks.

The methods introduced in this thesis should be considered as preliminary work. For example, the failure to synthesize motion with good quality should not be considered as a failure of the sequential Monte Carlo framework. Furthermore, though the efficiency of the sampling was made more efficient with the reference data, we expect further research to yield better results.

8.2 Future work

The ideas presented in this thesis can be further developed in many ways to address the limitations presented and to find new applications for the framework. As a conclusion for the thesis we present a couple of approaches for future work:

Accurate modeling of biomechanics. Other authors have improved the quality of their physically-based motion synthesis methods by more accurately modeling the human biomechanics. The simple model used in our system has room for improvement, and as the optimization process treats the simulation component as a black box, improvements should be easy to integrate into the system.

Perceptual motion quality measures. The error function used for reference animation tracking was constructed as the sum of squared differences for simplicity and mathematical convenience. Since the goal is to track animations in a way that looks natural to humans, a better alternative would be to model the perceptual error.

Better estimates for uncertainty in learning. The training data is gathered by following the reference animations as closely as possible. Thus, the training data systematically underestimates the variance in the possible motions, leading to poor generalization. Priors could be used to bias the learned model to higher variance, or the training data could be gathered specifically to find the true variance.

Learning from failures. The online optimizer could be used to generate training data: states where the existing training data leads to optimization failure could be found, and more optimizer samples could be used to generate solutions. In theory, using samples generated in this way, the system would learn increasingly robust control.

Physics simulation performance improvements. The implemented system integrates the Open Dynamics Engine (ODE), but there are other physics engines that should allow for quicker simulation and subsequently more samples to be generated. At the time of writing, the potentially faster Bullet physics engine is being integrated into the system.

Constrained online optimization strategies. In our system, the control signal is optimized frame-by-frame using the space of all motions, which sometimes causes the optimizer to pick a different motion for each frame. A more constrained alternative used by some authors is to pick one type of motion and to optimize online only to track that motion.

Per-segment online optimization. In the machine learning comparison, we used a version of mixture of regressors which was factorized as a sequence of spline segments. The same idea could be used in the optimizer by sampling one segment at a time instead of sampling and evaluating full splines.

Applications. The only application of the framework presented in this thesis was simple human locomotion - a feat that is certainly not unique in the field of motion synthesis. In the introduction, we briefly mentioned that the current state of motion synthesis is limiting the design of video games: we would like to see whether the technology advances have changed the situation.

Bibliography

[1] Abe, Y., da Silva, M., and Popović, J. Multiobjective control with frictional contacts. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2007), SCA '07, Eurographics Association, pp. 249–258.

[2] Agarwal, A., and Triggs, B. Monocular human motion capture with a mixture of regressors. In *In Proc. IEEE Workshop on Vision for Human-Computer Interaction* (2005).

[3] Akenine-Moller, T., Moller, T., and Haines, E. *Real-Time Rendering*, 2nd ed. A. K. Peters, Ltd., Natick, MA, USA, 2002.

[4] Al Borno, M., de Lasa, M., and Hertzmann, A. Trajectory optimization for full-body movements with complex contacts. *IEEE Transactions on Visualization and Computer Graphics 19*, 8 (Aug. 2013), 1405–1414.

[5] Anitescu, M., and Potra, F. A. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics 14* (1997), 231–247.

[6] Arthur, D., and Vassilvitskii, S. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (Philadelphia, PA, USA, 2007), SODA '07, Society for Industrial and Applied Mathematics, pp. 1027–1035.

[7] Baraff, D. Coping with friction for non-penetrating rigid body simulation. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1991), SIGGRAPH '91, ACM, pp. 31–41.

[8] Barber, D. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, New York, NY, USA, 2012.

[9] Bear, M. F., Connors, B., and Paradiso, M. *Neuroscience: Exploring the Brain (Third Edition)*, third ed. Lippincott Williams & Wilkins, Feb. 2006.

[10] Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[11] Bishop, C. M., Svensén, M., and Williams, C. K. I. GTM: The generative topographic mapping. *Neural Computation 10* (1998), 215–234.

[12] Breiman, L. Random forests. *Mach. Learn. 45*, 1 (Oct. 2001), 5–32.

[13] Cleveland, W. S. Robust locally weighted regression and smoothing scatterplots. *Journal of the American statistical association 74*, 368 (1979), 829–836.

[14] Cottle, R. W., and Dantzig, G. B. Complementary pivot theory of mathematical programming. *Linear Algebra and its Applications 1*, 1 (1968), 103 – 125.

[15] Da Silva, M., Abe, Y., and Popovic, J. Simulation of human motion data using short-horizon model-predictive control. *Comput. Graph. Forum 27*, 2 (2008), 371–380.

[16] Delp, S. L., Anderson, F. C., Arnold, A. S., Loan, P., Habib, A., John, C. T., Guendelman, E., and Thelen, D. G. Opensim: Open-source software to create and analyze dynamic simulations of movement. *IEEE Trans. Biomed. Engineering 54*, 11 (2007), 1940–1950.

[17] Dempster, A. P., Laird, N. M., and Rubin, D. B. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B 39* (1977), 1–38.

[18] Dorigo, M., Maniezzo, V., and Colorni, A. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B 26*, 1 (1996), 29–41.

[19] Doucet, A., and Johansen, A. M. A tutorial on particle filtering and smoothing: fifteen years later. In *Oxford Handbook of Nonlinear Filtering* (2011), pp. 656–704.

[20] Erez, T., Lowrey, K., Tassa, Y., Kumar, V., Kolev, S., and Todorov, E. An integrated system for real-time model-predictive control of humanoid robots. In *Proc. IEEE/RAS International Conference on Humanoid Robots (HUMANOIDS)* (2013), HUMANOIDS.

[21] Ericson, C. *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology) (The Morgan Kaufmann Series in Interactive 3D Technology)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[22] Geijtenbeek, T., and Pronost, N. Interactive character animation using simulated physics: A state-of-the-art review. *Comput. Graph. Forum 31*, 8 (Dec. 2012), 2492–2515.

[23] Geijtenbeek, T., Pronost, N., and van der Stappen, A. Simple Data-Driven Control for Simulated Bipeds. In *Eurographics/ACM SIGGRAPH Symposium on Computer Animation* (2012), pp. 211–219.

[24] Geijtenbeek, T., van de Panne, M., and van der Stappen, A. F. Flexible muscle-based locomotion for bipedal creatures. *ACM Trans. Graph. 32*, 6 (Nov. 2013), 206:1–206:11.

[25] Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[26] Gordon, N. J., Salmond, D. J., and Smith, A. F. M. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *Radar and Signal Processing, IEE Proceedings F 140*, 2 (Apr. 1993), 107–113.

[27] Han, D., Noh, J., Jin, X., and (formerly Sung Y. Shin), J. S. S. On-line real-time physics-based predictive motion control with balance recovery. *Comput. Graph. Forum 33*, 2 (2014), 245–254.

[28] Hansen, N. The CMA evolution strategy: a comparing review. In *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, J. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea, Eds. Springer, 2006, pp. 75–102.

[29] Haykin, S. *Neural Networks: A Comprehensive Foundation*, 2nd ed. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998.

[30] Heck, R., and Gleicher, M. Parametric motion graphs. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2007), I3D '07, ACM, pp. 129–136.

[31] Hämäläinen, P., Aila, T., Takala, T., and Al, J. Mutated kd-tree importance sampling. In *In Proceedings of the The Ninth Scandinavian Conference on Artificial Intelligence (SCAI 2006)* (2006).

[32] Hämäläinen, P., Eriksson, S., Tanskanen, E., Kyrki, V., and Lehtinen, J. Online motion synthesis using sequential Monte Carlo. *ACM Trans. Graph. 33*, 4 (July 2014), 51:1–51:12.

[33] Hotelling, H. Analysis of a complex of statistical variables into principal components. *J. Educ. Psych. 24* (1933).

[34] Ihler, A., Ihler, E. T., Sudderth, E., Freeman, W., and Willsky, A. Efficient multiscale sampling from products of gaussian mixtures. In *In NIPS 17* (2003), MIT Press, p. 2003.

[35] Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. Adaptive mixtures of local experts. *Neural Comput. 3*, 1 (Mar. 1991), 79–87.

[36] Jain, S., and Liu, C. K. Controlling physics-based characters using soft contacts. In *Proceedings of the 2011 SIGGRAPH Asia Conference* (New York, NY, USA, 2011), SA '11, ACM, pp. 163:1–163:10.

[37] Jain, S., Ye, Y., and Liu, C. K. Optimization-based interactive motion synthesis. *ACM Trans. Graph. 28*, 1 (2009), 1–10.

[38] Kavan, L., Collins, S., Žára, J., and O'Sullivan, C. Skinning with dual quaternions. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2007), I3D '07, ACM, pp. 39–46.

[39] Kohonen, T. Self-organized formation of topologically correct feature maps. *Biological cybernetics 43*, 1 (1982), 59–69.

[40] Kohonen, T. *Self-Organizing Maps*, 3rd ed. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

[41] Kovar, L., Gleicher, M., and Pighin, F. Motion graphs. *ACM Trans. Graph. 21*, 3 (July 2002), 473–482.

[42] Kovar, L., Schreiner, J., and Gleicher, M. Footskate cleanup for motion capture editing. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2002), SCA '02, ACM, pp. 97–104.

[43] Kreyszig, E. *Advanced Engineering Mathematics*, 8th ed. John Wiley & Sons, Inc., New York, NY, USA, 2000.

[44] Kwon, T., and Hodgins, J. Control systems for human running using an inverted pendulum model and a reference motion capture sequence. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2010), SCA '10, Eurographics Association, pp. 129–138.

[45] Lasseter, J. Principles of traditional animation applied to 3D computer animation. *SIGGRAPH Comput. Graph. 21*, 4 (Aug. 1987), 35–44.

[46] Levine, S., and Popovic, J. Physically plausible simulation for character animation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2012), SCA '12, Eurographics Association, pp. 221–230.

[47] Liu, L., Yin, K., van de Panne, M., and Guo, B. Terrain runner: control, parameterization, composition, and planning for highly dynamic motions. *ACM Trans. Graph. (TOG) 31*, 6 (2012), 154.

[48] Liu, L., Yin, K., van de Panne, M., Shao, T., and Xu, W. Sampling-based contact-rich motion control. *ACM Trans. Graph. 29*, 4 (2010), Article 128.

[49] Lloyd, S. Least squares quantization in PCM. *IEEE Trans. Inf. Theor. 28*, 2 (Sept. 2006), 129–137.

[50] Mockus, J. *Bayesian approach to global optimization: theory and applications*. Mathematics and its applications (Kluwer Academic Publishers).: Soviet series. Kluwer Academic, 1989.

[51] Mordatch, I., de Lasa, M., and Hertzmann, A. Robust physics-based locomotion using low-dimensional planning. In *ACM SIGGRAPH 2010 Papers* (New York, NY, USA, 2010), SIGGRAPH '10, ACM, pp. 71:1–71:8.

[52] Mordatch, I., Todorov, E., and Popović, Z. Discovery of complex behaviors through contact-invariant optimization. *ACM Trans. Graph. 31*, 4 (July 2012), 43:1–43:8.

[53] Muico, U., Lee, Y., Popović, J., and Popović, Z. Contact-aware nonlinear control of dynamic characters. *ACM Trans. Graph. 28*, 3 (2009).

[54] Muico, U., Popović, J., and Popović, Z. Composite control of physically simulated characters. *ACM Trans. Graph. 30*, 3 (2011).

[55] Muja, M., and Lowe, D. G. Scalable nearest neighbor algorithms for high dimensional data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 36* (2014).

[56] Pearson, K. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine 2*, 6 (1901), 559–572.

[57] Pejsa, T., and Pandzic, I. S. State of the art in example-based motion synthesis for virtual characters in interactive applications. *Comput. Graph. Forum 29*, 1 (2010), 202–226.

[58] Petersen, K. B., and Pedersen, M. S. The matrix cookbook. *Technical University of Denmark* (2008), 7–15.

[59] Pharr, M., and Humphreys, G. *Physically Based Rendering, Second Edition: From Theory To Implementation*, 2nd ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010.

[60] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, 3 ed. Cambridge University Press, New York, NY, USA, 2007.

[61] Safonova, A., and Hodgins, J. K. Analyzing the physical correctness of interpolated human motion. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2005), SCA '05, ACM, pp. 171–180.

[62] Safonova, A., Hodgins, J. K., and Pollard, N. S. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. In *ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), SIGGRAPH '04, ACM, pp. 514–521.

[63] Sharon, D., and van de Panne, M. Synthesis of controllers for stylized planar bipedal walking. In *ICRA* (2005), pp. 2387–2392.

[64] Sims, K. Evolving virtual creatures. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1994), SIGGRAPH '94, ACM, pp. 15–22.

[65] Sok, K. W., Kim, M., and Lee, J. Simulating biped behaviors from human motion data. *ACM Trans. Graph. 26*, 3 (July 2007).

[66] Tang, J. K. T., Leung, H., Komura, T., and Shum, H. P. H. Emulating human perception of motion similarity. *Comput. Animat. Virtual Worlds 19*, 3-4 (Sept. 2008), 211–221.

[67] Tassa, Y., Erez, T., and Todorov, E. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on* (Oct 2012), pp. 4906–4913.

[68] Thayananthan, A., Navaratnam, R., Stenger, B., Torr, P. H. S., and Cipolla, R. Multivariate relevance vector machines for tracking. In *Proceedings of the 9th European Conference on Computer Vision - Volume Part III* (Berlin, Heidelberg, 2006), ECCV'06, Springer-Verlag, pp. 124–138.

[69] Tipping, M. E. Sparse bayesian learning and the relevance vector machine. *J. Mach. Learn. Res. 1* (Sept. 2001), 211–244.

[70] Wang, J. M., Hamner, S. R., Delp, S. L., and Koltun, V. Optimizing locomotion controllers using biologically-based actuators and objectives. *ACM Trans. Graph. 31*, 4 (July 2012), 25:1–25:11.

[71] Williams, R. *The Animator's Survival Kit–Revised Edition: A Manual of Methods, Principles and Formulas for Classical, Computer, Games, Stop Motion and Internet Animators.* Faber & Faber, Inc., 2009.

[72] Witkin, A., and Kass, M. Spacetime constraints. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1988), SIGGRAPH '88, ACM, pp. 159–168.

[73] Wu, J.-c., and Popovic, Z. Terrain-adaptive bipedal locomotion control. *ACM Trans. Graph. 29*, 4 (Jul. 2010), 72:1–72:10.

[74] Yin, K., Loken, K., and van de Panne, M. Simbicon: Simple biped locomotion control. In *ACM SIGGRAPH 2007 Papers* (New York, NY, USA, 2007), SIGGRAPH '07, ACM.

[75] Zordan, V. B., Majkowska, A., Chiu, B., and Fast, M. Dynamic response for motion capture animation. *ACM Trans. Graph. 24*, 3 (July 2005), 697–701.